ВВЕДЕНИЕ В ВЕБ РАЗРАБОТКУ С HTML, CSS, JAVASCRIPT

ТИМУР МАШНИН

Тимур Машнин

Введение в веб-разработку с HTML, CSS, JavaScript

«Автор»

2023

Машнин Т.

Введение в веб-разработку с HTML, CSS, JavaScript /
Т. Машнин — «Автор», 2023

Изучите основы веб-разработки, создавая веб-страницы с помощью HTML, CSS и JavaScript. С этой книгой Вы научитесь использовать HTML для создания веб-страницы с абзацами, divs, изображениями, ссылками и списками, добавлять стили на веб-страницу с помощью идентификаторов и классов CSS, делать веб-страницу интерактивной с помощью команд JavaScript. Вы узнаете как создать веб-страницу, которая будет так же удобна на мобильном телефоне, как и на настольном компьютере. Научитесь работать с объектной моделью документа DOM и узнаете что такое фронтенд и бэкенд разработка.

Тимур Машнин
Введение в веб-разработку с HTML, CSS, JavaScript

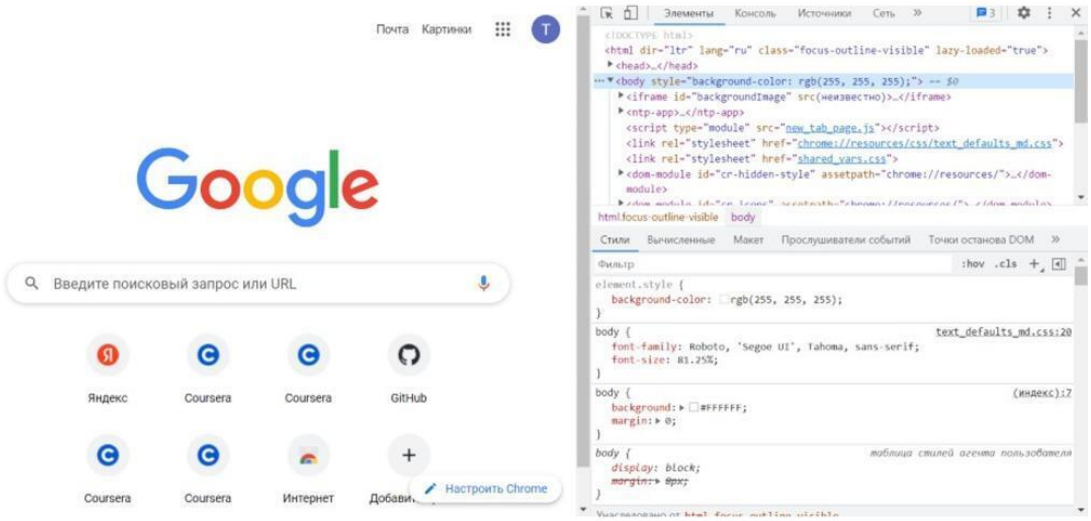
Начало работы

Для начала работы с веб-проектами необходимо настроить среду разработки.

Какие инструменты составляют среду разработки?

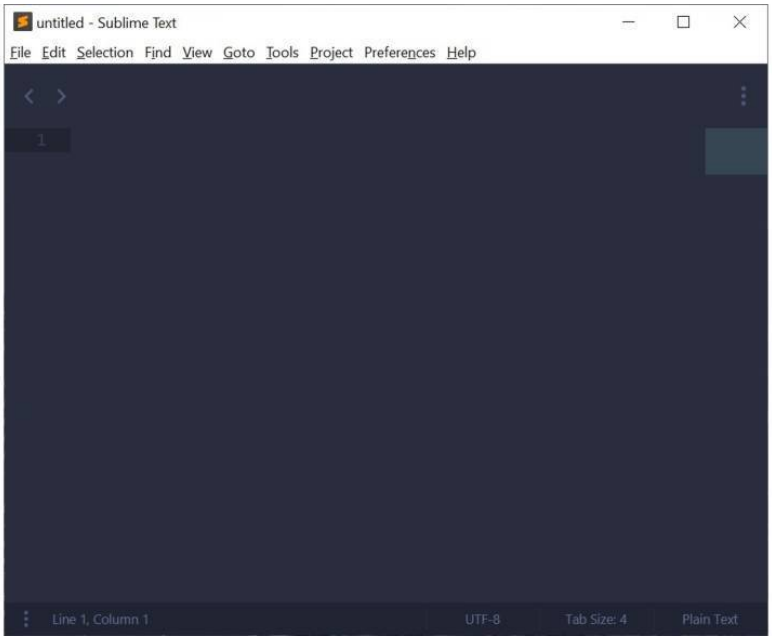
Во-первых, для веб-разработки можно использовать браузер Google Chrome.

Браузер Google Chrome уже поставляется с инструментами CDT, Chrome Developer Tools.



Далее нам нужен редактор кода.

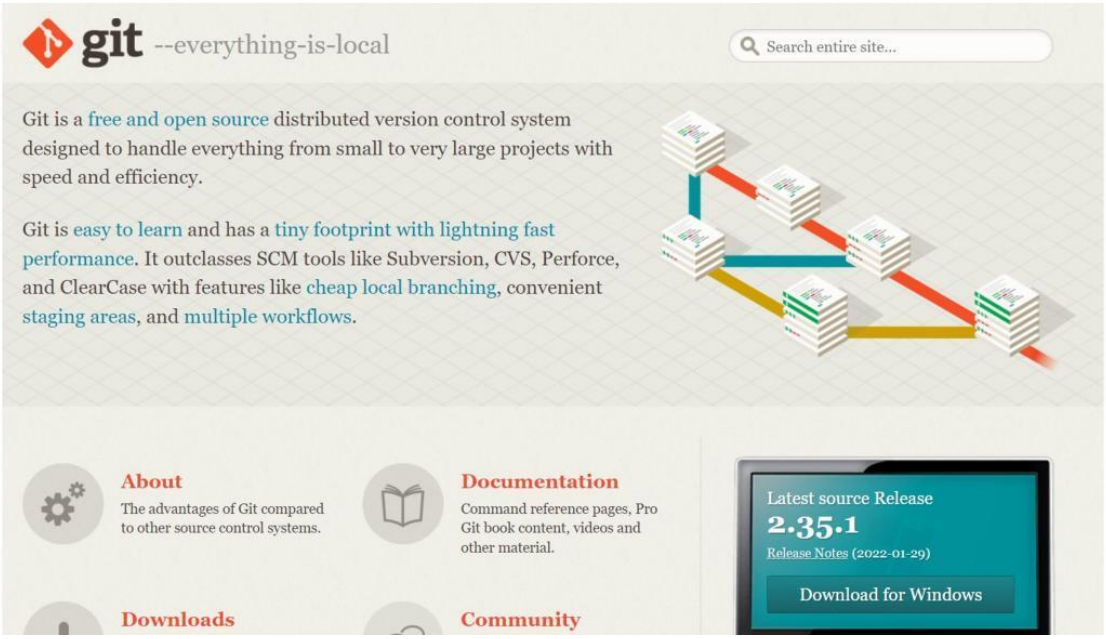
И в качестве редактора кода можно использовать редактор Sublime Text 3.



Sublime чрезвычайно быстр и очень хорошо поддерживается с точки зрения сторонних плагинов.

Для установки редактора Sublime в интернете можно найти дитрибутив в том числе и портативной версии.

Нам также понадобится Git.

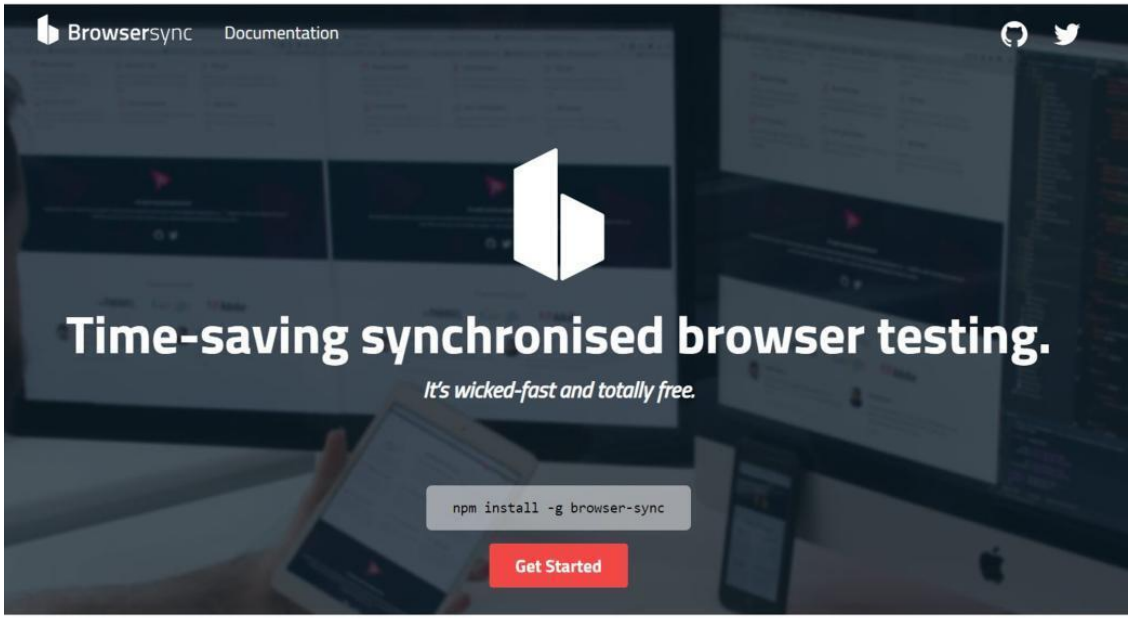


Git – это система управления версиями, с помощью которой можно хранить информацию о всех изменениях в вашем коде в репозитории.

Репозиторий Git – это место, где хранится ваш код и вся информация о его изменениях.

Репозитории могут находиться локально у вас на компьютере или на удалённом компьютере.

Для установки Git скачаем и запустим дистрибутив, и далее будем следовать инструкциям.



Также мы можем использовать инструмент под названием Browser Sync.

Browsersync – это модуль Node.js.

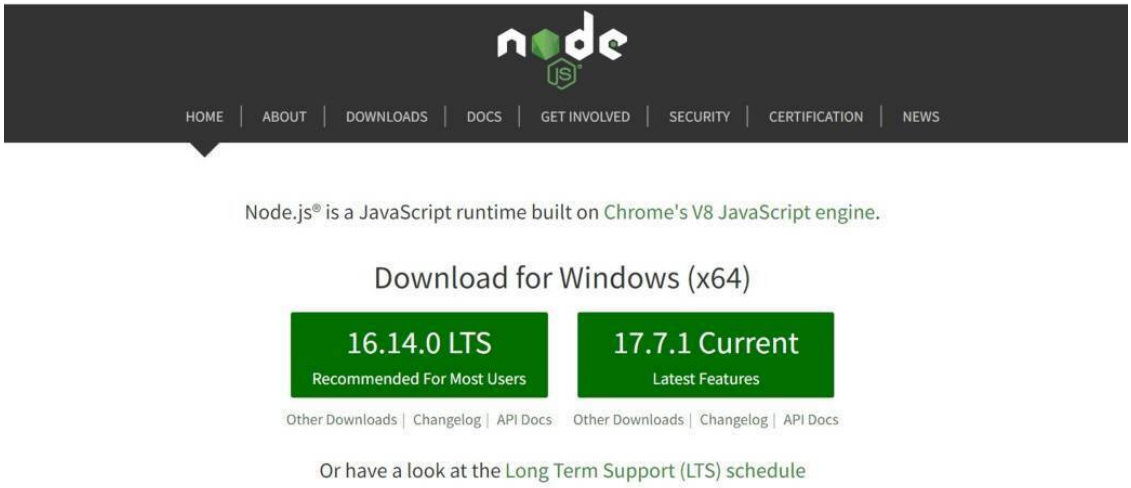
Node.js – это среда выполнения приложений JavaScript, которая выполняет код JavaScript вне веб-браузера.

И Browsersync обеспечивает синхронизированное тестирование веб приложений в браузере.

Browsersync создает сервер по адресу http://localhost:3000/ и наблюдает за изменениями файлов веб приложения. И когда файлы изменяются, Browsersync автоматически перезагружает браузер и мгновенно показывает эти изменения.

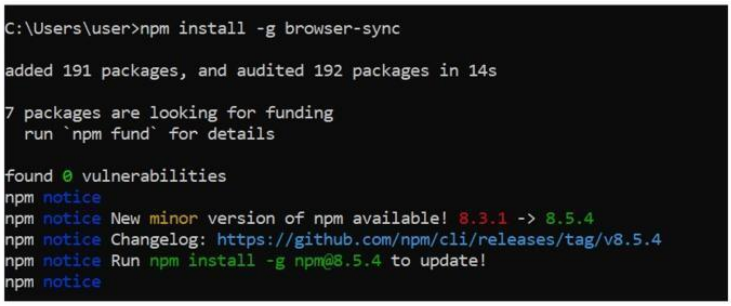
Таким образом экономится наше время на обновление браузера во время разработки веб приложения.

И чтобы установить Browser Sync, нам нужно установить Node.js.



Для этого скачаем и запустим дистрибутив Node.js, и далее будем следовать инструкциям.

```
npm install -g browser-sync
```



Далее установим Browsersync.

Если вы хотите запускать Browsersync из командной строки в любом каталоге, его можно установить глобально с помощью выше показанной команды в терминале командной строки.

Здесь NPM (Node Package Manager) – это менеджер пакетов Node.js. И NPM устанавливается вместе с Node.js.

```
browser-sync start -- server --files "index. html“

browser-sync start --server 'app' --files 'app’

browser-sync start --directory --server --files "*"
```

Теперь можно запускать файлы проекта веб-приложения с помощью Browsersync.

Для этого нужно открыть терминал и ввести выше показанную команду.

Эта команда запускает локальный сервер по адресу http://localhost:3000, открывает и проверяет изменения файла index.html.

Можно также запустить сервер Browsersync для каталога приложения, отслеживая все его файлы.

Введение в HTML

Итак, что же такое HTML? Что вообще означает термин HTML и как он связан с Интернетом?

HTML – это язык гипертекстовой разметки. И давайте рассмотрим каждое из этих слов и выясним, что именно они означают.

Во-первых, гипертекст. И гипертекст – это текст, который содержит ссылки на другие тексты, и так, по сути, устроена вся сеть Интернет. Один документ указывает на другой документ, который указывает на кучу других документов, и так далее.

```
<!DOCTYPE HTML>
<html>
<head>
<title>NOV Tech Solutions</title>
<meta name="description" content="NOV Tech Solutions - создание динамических сайтов, мобильных приложений, автоматизация бизнес-процессов">
<meta http-equiv="content-type" content="text/html;charset=UTF-8">
<link href="css/bootstrap.css" rel='stylesheet' type='text/css' />
<script src="js/jquery.min.js"></script>
<script type="text/javascript" src="js/move-top.js"></script>
<script type="text/javascript" src="js/easing.js"></script>
        <link href="css/theme-style.css" rel='stylesheet' type='text/css' />
        <meta name="viewport" content="width=device-width, initial-scale=1">
        <script type="application/x-javascript"> addEventListener("load", function() { setTimeout(hideURLbar, 0); }, false); function hideURLbar(){ window.scrollTo(0,1); }
        </script>
        <link href='http://fonts.googleapis.com/css?family=Open+Sans:400,300,600,700,800' rel='stylesheet' type='text/css'>
        <script>
            $(function() {
                var pull        = $('#pull');
                    menu        = $('nav ul');
                    menuHeight  = menu.height();
                $(pull).on('click', function(e) {
                    e.preventDefault();
                    menu.slideToggle();
                });
                $(window).resize(function(){
                    var w = $(window).width();
                    if(w > 320 && menu.is(':hidden')) {
                        menu.removeAttr('style');
                    }
                });
            });
        </script>
</head>
    <body>
```

Следующий термин – разметка. И разметка означает пометить что-то, аннотировать, т.е. добавить аннотацию.

Например, если у вас есть какой-то контент, например, если у вас есть HTML документ, и вы хотите аннотировать его, чтобы сообщить браузеру, что это за контент и что делает этот HTML документ, вы обертываете этот контент в язык разметки, состоящий из тегов.

Например, тег title, заголовок, сообщает нам и другому программному обеспечению, что это название этого документа.

Таким образом, теги описывают структуру документа.

И последнее слово – это язык, а язык подразумевает, что у него есть собственный синтаксис, означающий, что есть правильный и неправильный способ кодирования.

Так, например, в этом примере закрывающий тег div появляется после закрывающего тега h1, что является синтаксической ошибкой.

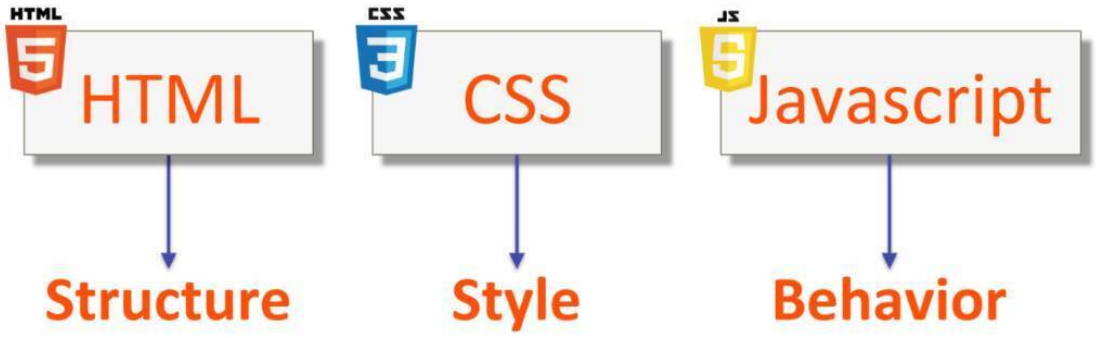
Теперь, на самом деле Интернет основывается на трех базовых технологиях HTML, CSS, и JavaScript.

HTML, как мы уже выяснили, описывает структуру документа.

Например, документ может иметь один заголовок, два абзаца и нижний колонтитул.

И обратите внимание, что это ничего не говорит о том, как эти компоненты визуально расположены, как они выглядят, какого они цвета и какой при этом размер шрифта.

Цвет и стиль – это роль CSS.



Язык Cascading Style Sheets (CSS) определяет цвет, макет, стиль шрифта, размер шрифта, другими словами стиль документа.

И кроме HTML и CSS, есть еще язык JavaScript, работа которого заключается в обеспечении поведения или функциональности документа.

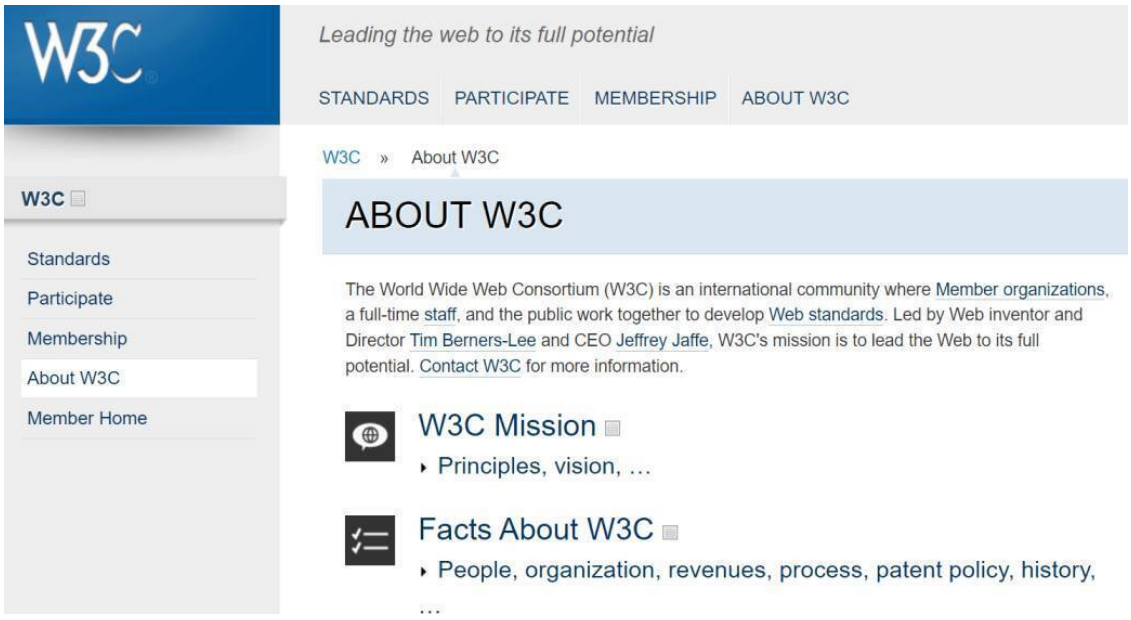
JavaScript добавляет функциональность странице, например, что происходит, когда пользователь нажимает на заголовок документа.

Теперь, как появился HTML?

До 1997 года не существовало никаких стандартов Интернета, и браузеры делали все, что хотели. Они изобретали новые теги и по-разному реализовывали одни и те же теги.

И вы могли зайти на веб-сайт и получить сообщение о том, что ваш браузер несовместим с этим веб-сайтом, поэтому вам нужно перейти на другой браузер.

И примерно в 1997 году Консорциум World Wide Web, W3C, придумал первый стандарт для браузеров, это был HTML4, и они очень быстро обновили его до HTML4.01.



Этот стандарт был довольно свободным, и у браузеров было слишком много свободы действий внутри этого стандарта в отношении того, как его реализовывать и как отображать страницы.

Поэтому, примерно в 2000 году W3C предложил другую спецификацию под названием XHTML 1.0, и эта спецификация была основана на языке разметки XML. И W3C хотели продолжить работу, создав XHTML 2.0.

Но проблема была в том, что поставщики браузеров решили, что работа W3C продвигается слишком медленно, и, кроме того, они считали, что спецификации W3C движутся в неправильном направлении.

Поэтому, производители браузеров объединились и создали еще одну группу разработки спецификаций, которая называлась WHATWG, группа технологий веб-гипертекстовых приложений.

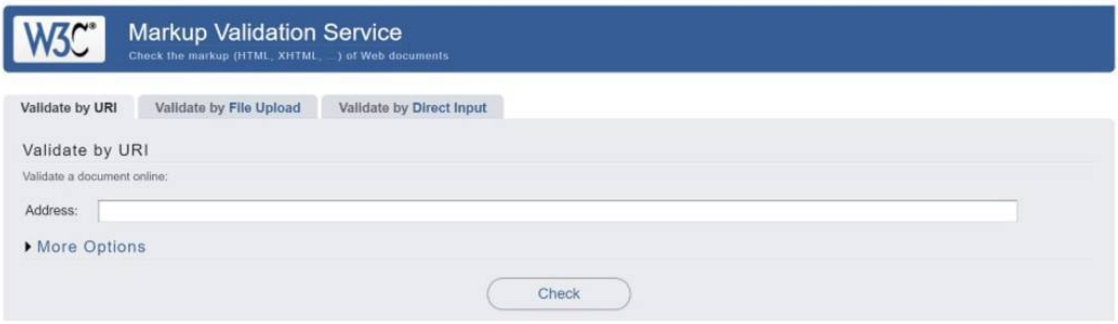
И эта группа была гораздо менее демократична, чем W3C, так как там есть один главный редактор, который принимает окончательные решения, поэтому все представители поставщиков браузеров могут спорить сколько угодно, но в конце концов назначенный редактор принимает окончательные решения.

И в течение долгого времени эти две организации не сходились во взглядах и не работали вместе, так что они действительно шли в двух разных направлениях. Но в конце концов, примерно в 2007, 2009 годах, WHATWG и W3C начали совместную работу, и создали то, что мы имеем сейчас, стандарт HTML5.

Теперь, W3C отвечает за стандарты, а WHATWG отвечает за реализацию стандартов браузерами.

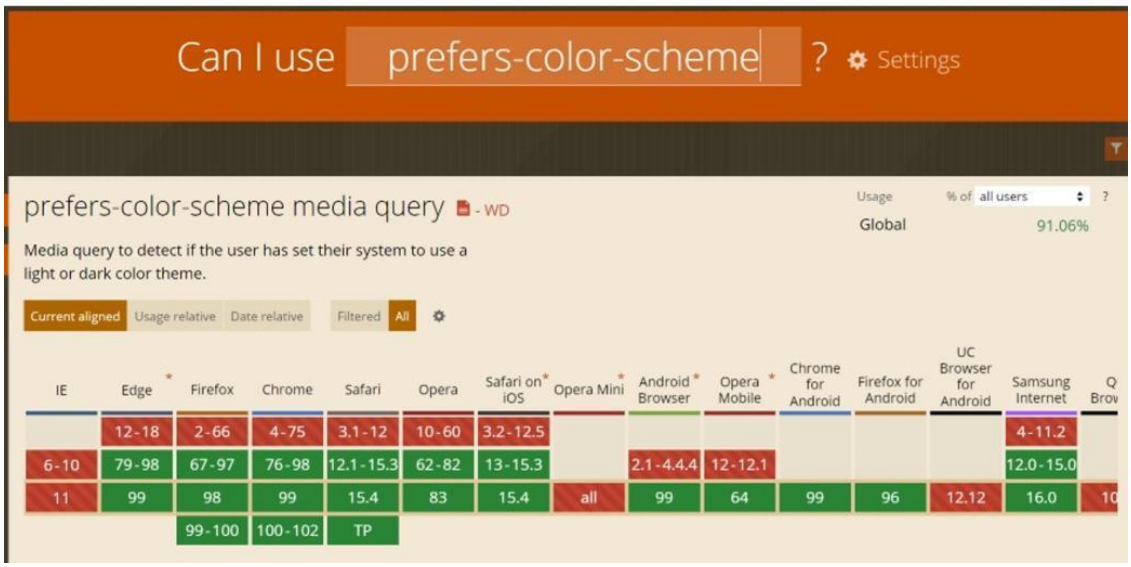
И чтобы узнать, будет ли ваш HTML работать в браузерах, вы можете использовать валидатор, который есть на веб-сайте W3.org.

https://validator.w3.org/



Здесь вы можете проверить свой код HTML, является ли он действительным и будет ли он хорошо работать в браузере.

https://caniuse.com/



С помощью сайта CanIUse вы можете проверить конкретную HTML функцию, совместима ли она с конкретными браузерами, что может помочь вам легче принимать решения по дизайну и разработке.

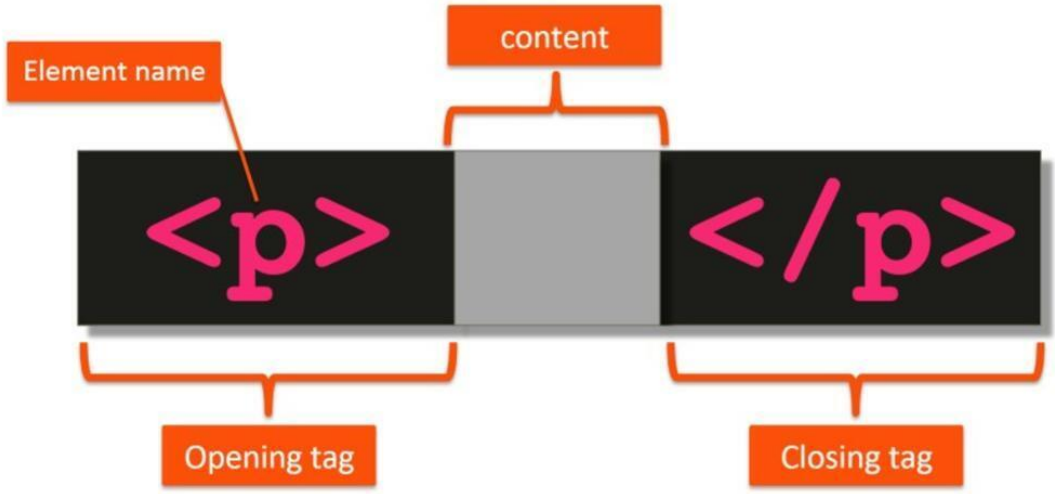
Здесь таблицы совместимости указывают не только на то, поддерживает ли конкретный браузер технологию или нет, но и в какой степени, на случай частичной совместимости в определенных версиях.

При этом узнать долю использования браузера в Интернете вы можете с помощью сайта w3schools.com.

https://www.w3schools.com/browsers/

2022	Chrome	Edge	Firefox	Safari	Opera
February	79.9 %	7.5 %	5.4 %	4.0 %	2.3 %
January	80.1 %	7.3 %	5.5 %	3.9 %	2.3 %
2021	**Chrome**	**Edge**	**Firefox**	**Safari**	**Opera**
December	81.0 %	6.6 %	5.5 %	3.7 %	2.3 %
November	80.0 %	6.8 %	5.8 %	3.9 %	2.4 %
October	80.3 %	6.7 %	5.7 %	3.9 %	2.3 %
September	80.9 %	6.5 %	5.6 %	3.6 %	2.2 %
August	81.4 %	6.1 %	5.6 %	3.3 %	2.1 %
July	81.6 %	6.0 %	5.6 %	3.3 %	2.2 %
June	81.7 %	5.9 %	5.6 %	3.4 %	2.2 %
May	81.2 %	5.8 %	5.8 %	3.5 %	2.4 %
April	80.7 %	5.6 %	6.1 %	3.7 %	2.4 %

В основе HTML лежат теги HTML. Поэтому очень важно понимать, из чего состоит HTML-тег и как правильно, синтаксически правильно кодировать HTML-тег.



Как правило, теги HTML имеют открывающий и закрывающий тег. И они окружают какой-то контент.

В данном случае, тег p, обозначающий абзац, сообщает нам, что содержимое в середине следует рассматривать как абзац.

С технической точки зрения, сам по себе p называется элементом. И вместе с угловыми скобками это называется тегом.

И у большинства HTML-тегов есть закрывающий тег, который совпадает с открывающим тегом, но не у всех.

```
<p>To force<br> line breaks<br> in a text,<br> use the br<br> element.
</p>
```

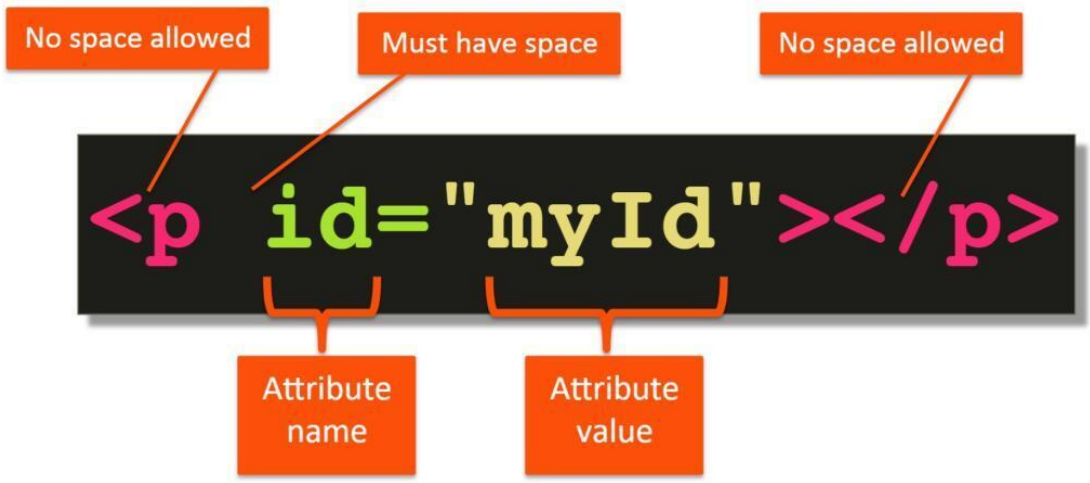
To force
line breaks
in a text,
use the br
element.

```
<p>HTML is the standard markup language for creating Web pages. HTML
describes the structure of a Web page, and consists of a series of
elements. HTML elements tell the browser how to display the content.
</p>

<hr>
```

HTML is the standard markup language for creating Web pages. HTML describes the structure of a Web page, and consists of a series of elements. HTML elements tell the browser how to display the content.

Например, теги br и hr, br означает разрыв строки, а hr означает горизонтальную разделительную линию, имеют только открывающий тег. У них вообще нет закрывающего тега.



Теперь, каждый элемент HTML может иметь предопределенные атрибуты.

Атрибут – это пара «имя-значение», представляющая собой своего рода метаданные о самом элементе, к которому он применяется.

(Метаданные – информация о другой информации, или данные, относящиеся к дополнительной информации о содержимом или объекте.)

В этом примере мы присваиваем myId как значение атрибута id, который представляет идентификатор элемента HTML.

И каждый атрибут имеет свои собственные правила для присваивания его значения.

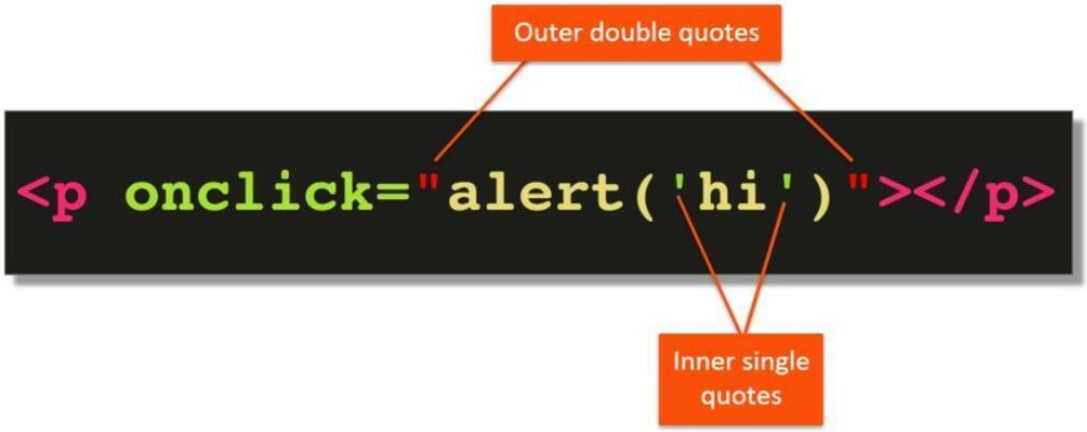
Так, например, значение атрибута id должно быть уникальным в рамках всего HTML-документа. Если есть другой элемент с тем же значением для id, это будет означать, что веб-страница содержит недопустимый HTML-код.

Что касается синтаксиса, между открывающей скобкой и именем тега не должно быть пробелов. Точно так же не допускается пробел между открывающей скобкой и косой чертой перед закрывающим тегом. Однако у вас должен быть хотя бы один пробел между самим тегом и любым из его атрибутов.

И еще одно правило – атрибуты могут быть указаны только в открывающем теге, поэтому вы не можете указать атрибут в закрывающем теге.

Также рекомендуется всегда заключать значение атрибута в одинарные или двойные кавычки. Неважно, используете ли вы одинарные или двойные кавычки, так как они эквивалентны в HTML.

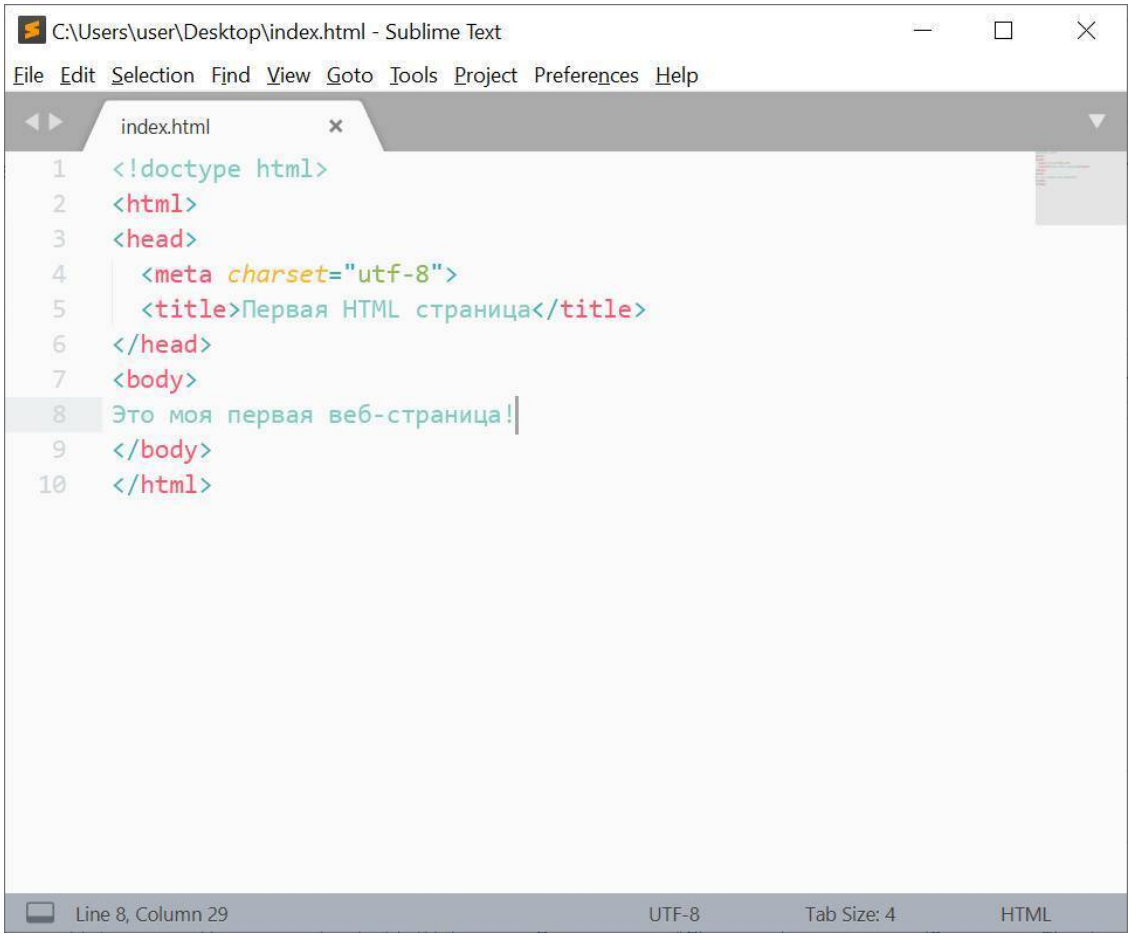
Более интересный случай возникает, когда само значение атрибута содержит кавычки. В такой ситуации рекомендуется использовать и одинарные и двойные кавычки.



Вы можете начать с двойных кавычек, или вы можете начать с одинарных кавычек.

Создание первой HTML страницы

Для создания кода HTML страницы откроем редактор Sublime.



И введем вышепоказанный код.

И каждая HTML-страница должна начинаться с объявления типа документа. То есть все HTML-документы должны начинаться с объявления <!DOCTYPE>, которое не является тегом HTML. Это «информация» для браузера о том, какой тип документа браузеру ожидать.

Тип документа – это какой спецификации HTML следует код страницы.

Если код страницы следует спецификации HTML 5, объявление типа документа простое – <!doctype html>.

Для HTML 4.01, такое объявление было гораздо сложнее:

<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" "http://www.w3.org/TR/html4/loose.dtd">

Слово doctype или HTML может быть написано строчными или прописными буквами. Единственное, на что вы должны обратить внимание, это то, что между восклицательным знаком и словом doctype не должно быть пробела.

И если вы пропустите объявление типа HTML-страницы, это будет сигналом для браузера, что он должен рассматривать ваши страницы как страницы, не соответствующие стандарту HTML.

Поэтому ваш код HTML может работать не совсем правильно.

Далее идет тег html, и это тег, внутри которого содержится весь html-документ.

После тега html идет тег head. И тег head содержит элементы, относящиеся к описанию содержимого страницы.

Например, какую кодировку символов должен использовать браузер для отображения контента страницы.

Тег head также может содержать заголовок страницы, это тег title, и ссылки на любые ресурсы, необходимые для правильного отображения страницы.

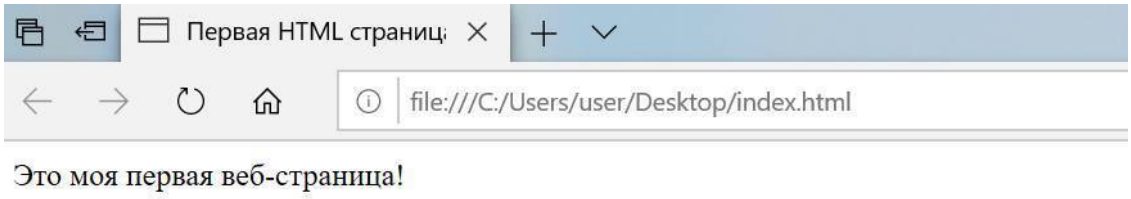
Таким образом, тег head содержит метаданные об основном контенте страницы.

Здесь тег meta с атрибутом charset указывает какую кодировку необходимо использовать для отображения символов нашей веб-страницы.

UTF-8 – это наиболее часто используемый стандарт кодирования символов, позволяющий компактно хранить и передавать символы Юникода.

И обратите внимание, что у тега meta нет закрывающего тега.

Далее мы указываем заголовок страницы с помощью тега title. И заголовок – это один из обязательных тегов, без которого HTML страница будет недействительная.



И заголовок отображается во вкладке браузера.

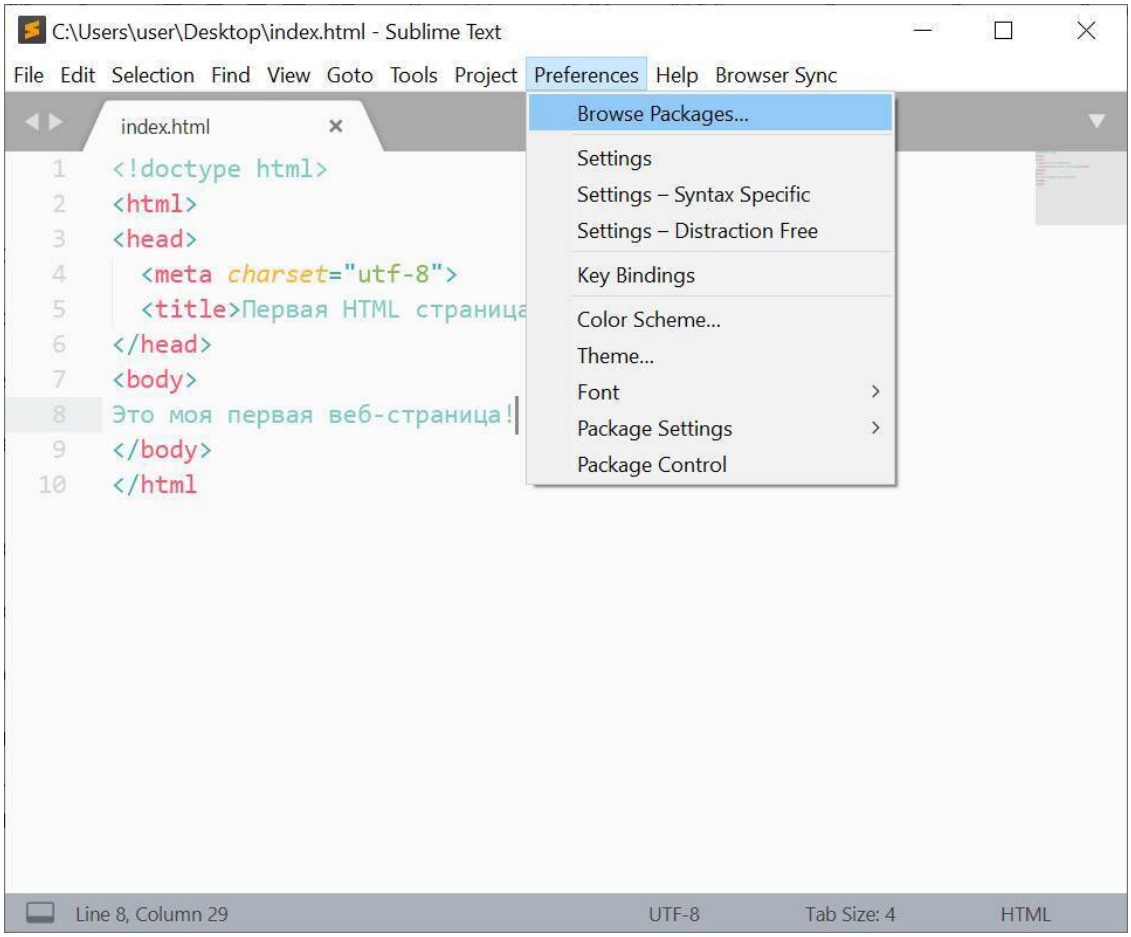
После тега head идет тег body. Тег body содержит весь основной контент, видимый пользователю.

Сохраним нашу страницу как index.html, так, как правило, называется главная страница сайта. И теперь давайте посмотрим, как это выглядит в браузере.

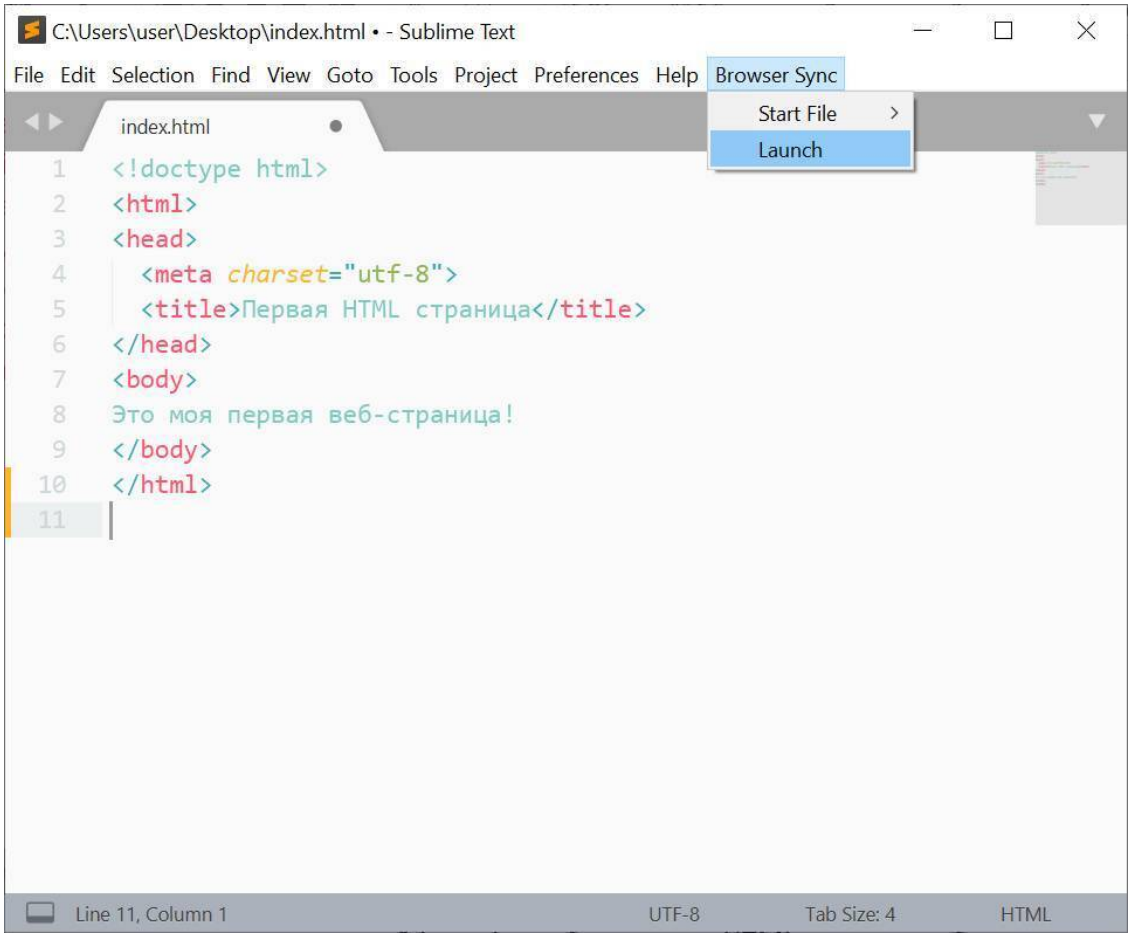
Но сначала установим плагин Browsersync для Sublime Text.

Для этого скачаем Github репозиторий проекта Browsersync по адресу https://github.com/iamdjones/sublime-text-browser-sync.

Распакуем папку и сохраним ее в каталоге пакетов Sublime Text, который мы найдем с помощью меню Preferences – Browse Packages.



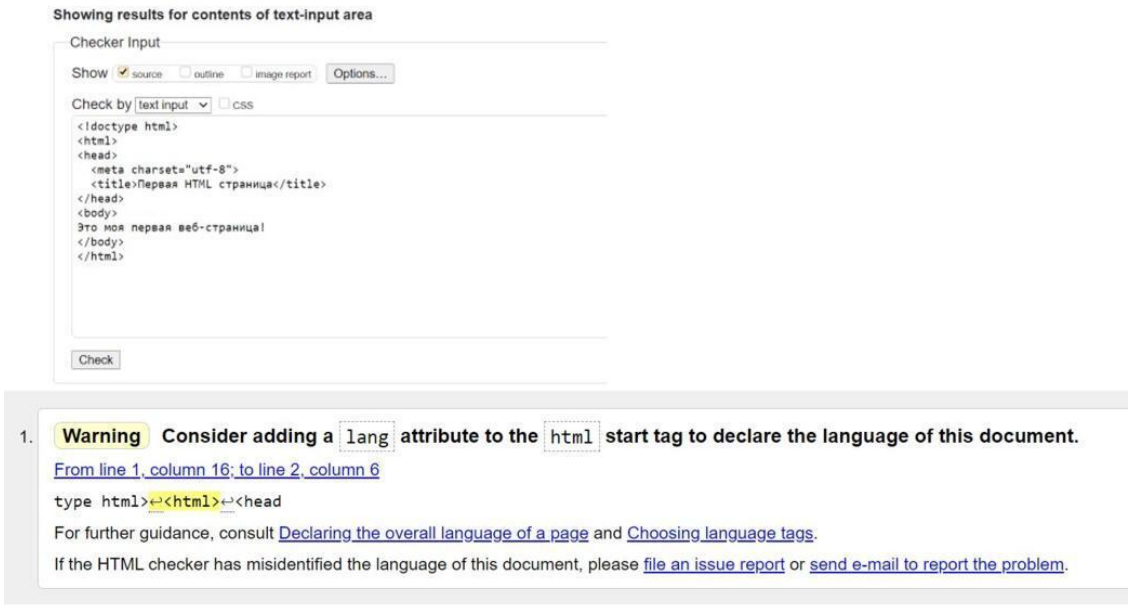
В результате у нас появится пункт меню Browser Sync.



Теперь мы можем воспользоваться командой Launch, чтобы открыть HTML страницу в браузере.

При этом все изменения кода страницы будут автоматически отображаться в браузере.

Далее давайте попробуем взять код нашей страницы, скопировать и вставить его для проверки внутри валидатора W3C.



В результате мы получим предупреждение, что у нас отсутствует атрибут lang тега html, который сообщает язык текстового содержимого. Эта информация помогает поисковым системам возвращать результаты для конкретного языка, а также используется программами чтения с экрана, которые переключают языковые профили для обеспечения правильного акцента и произношения.

Поэтому добавим этот атрибут – <html lang="en">.

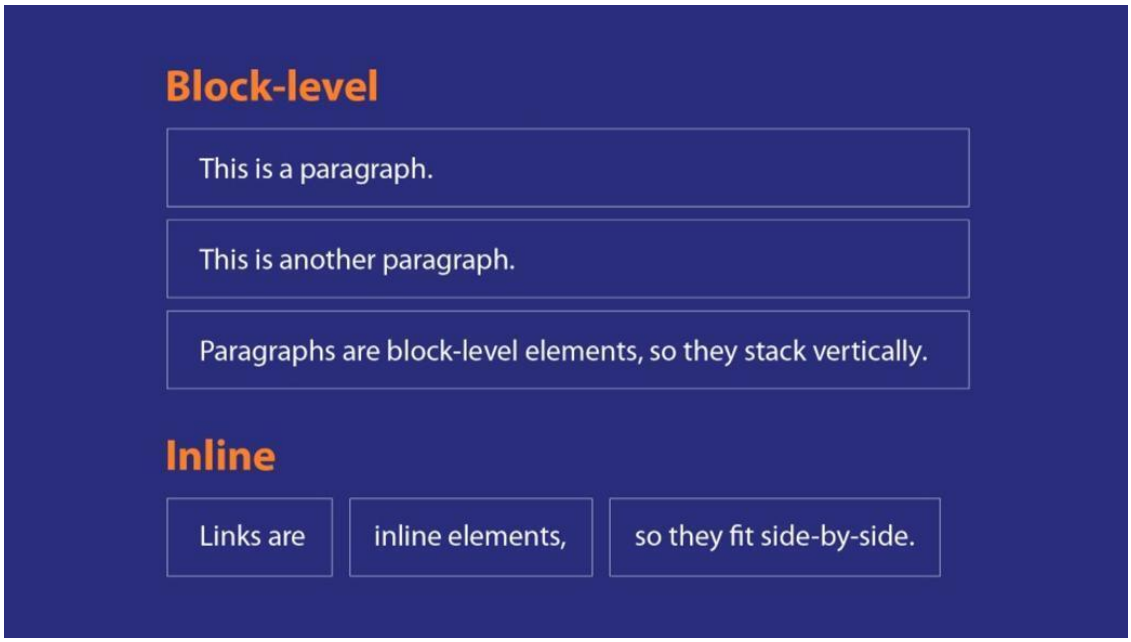
Теперь проверка в валидаторе покажет, что наша страница действительна.

Еще одно замечание, когда браузер открывает HTML-страницу, он всегда отображает или интерпретирует HTML-код последовательно сверху вниз.

Таким образом, сначала интерпретируется объявление типа документа, затем тег HTML, затем тег заголовка и так далее, пока не будет достигнут последний закрывающий тег HTML.

Элементы HTML документа

Все элементы HTML документа делятся на две категории в рамках традиционной HTML структуры. Это либо элементы уровня блока (block-level), либо встроенные элементы (inline).



Элементы уровня блока по умолчанию отображаются с новой строки. Вы можете изменить это с помощью CSS, но мы пока не говорим о CSS.

Это означает, что каждый раз, когда вы указываете блочный элемент в HTML, браузер автоматически помещает этот элемент на новую строку в потоке отображения документа.

И элементы уровня блока могут содержать внутри себя встроенные или другие элементы уровня блока. При этом встроенные элементы по умолчанию отображаются в одной строке. Опять же, вы можете изменить это, но по умолчанию они отображаются в одной и той же строке.

У встроенных элементов также есть ограничение, заключающееся в том, что они могут содержать только другие строчные элементы. Другими словами, встроенный элемент не может иметь как часть своего содержимого элемент уровня блока.

```
<!doctype html>
<html>
<head>
 <meta charset="utf-8">
 <title>div and span elements</title>
</head>
<body>
 <div>*** DIV 1: Some content here ***</div>
 <div>*** DIV 2: Following right after div 1 ***</div>
 <span>*** SPAN 1: Following right after div 2 ***</span>
 <div>
  *** DIV 3: Following right after span 1
  <span>*** SPAN 2: INSIDE div 3 ***</span>
  Continue content of div 3 ***
 </div>
</body>
</html>
```

```
*** DIV 1: Some content here ***
*** DIV 2: Following right after div 1 ***
*** SPAN 1: Following right after div 2 ***
*** DIV 3: Following right after span 1 *** SPAN 2: INSIDE div 3 *** Continue content of div 3 ***
```

В этом примере используются элементы div и span. Элемент div обозначает деление, а элемент span – интервал.

Элемент div – это самый общий элемент блочного уровня, а span – это суперуниверсальный встроенный элемент.

Здесь у нас есть пара элементов div, следующих один за другим, DIV 1 и DIV 2. Затем есть элемент span, который следует за DIV 2. И DIV 3 немного сложнее, так как он включает в себя элемент span внутри него.

Если открыть этот документ в браузере, вы можете увидеть, что элемент DIV 1 находится сам по себе на отдельной строке. Так же и элемент DIV 2, сам по себе на отдельной строке.

Элемент Span 1 следует сразу после DIV 2. И, хотя span является встроенным элементом, но поскольку DIV 2 располагается на отдельной строке, следующий элемент также размещается на отдельной строке.

DIV 3 является элементом уровня блока, поэтому он перемещается на следующую строку, но элемент Span 2 находится внутри DIV 3, и, поскольку это встроенный элемент, он не требует новой строки.

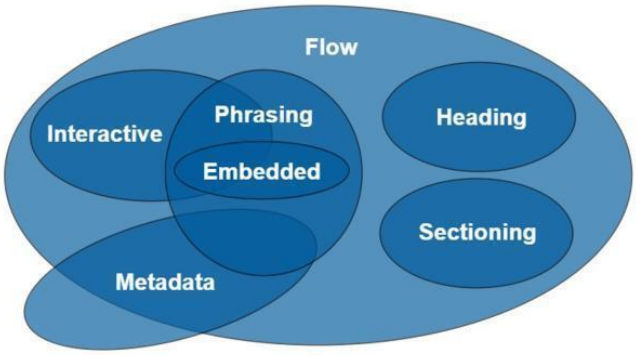
И обратите внимание, то, как мы пишем код и размещаем его на разных строках абсолютно не влияет на html-страницу и на то, как она отображается. Вы могли бы написать весь код в одну строку и результат будет тот же.

Теперь возьмите этот код, скопируйте его и проверьте в валидаторе W3C. Вы увидите, что страница действительна.

Но что произойдет, если вы прямо внутри валидатора добавите еще один тег div прямо внутри тега span с некоторым содержимым. Вы увидите, как валидатор пожалуется, что недопустимый элемент div является дочерним элементом span.

https://www.w3.org/TR/2011/WD-html5-20110525/contentmodels.html

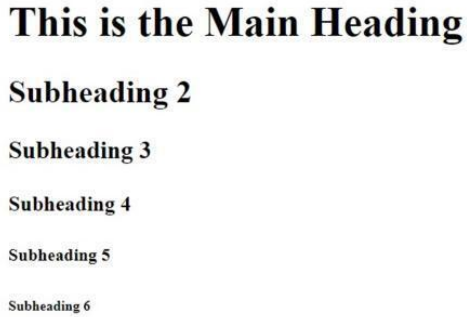
- Metadata content
- Flow content
- Sectioning content
- Heading content
- Phrasing content
- Embedded content
- Interactive content



На самом деле спецификация HTML5 группирует традиционные блочные и встроенные элементы в семь типов HTML контента. И вы можете посмотреть это разделение более подробно в разделе видов контента W3C, где перечислены семь типов контента, которые определяет HTML5.

Теперь, давайте рассмотрим следующие HTML элементы, представляющие заголовки документа.

```
<!doctype html>
<html>
<head>
  <meta charset="utf-8">
  <title>Heading Elements</title>
</head>
<body>
  <h1>This is the Main Heading</h1>
  <h2>Subheading 2</h2>
  <h3>Subheading 3</h3>
  <h4>Subheading 4</h4>
  <h5>Subheading 5</h5>
  <h6>Subheading 6</h6>
</body>
</html>
```



В этом очень простом HTML-документе есть элементы h1, h1, h2, h3 и вплоть до элемента h6, в общем, все доступные элементы заголовков.

И эти элементы указывают то, что содержимое заголовка между открывающим элементом h1 и закрывающим элементом h1 является самым важным заголовком в документе, и так далее.

Таким образом, h6 также будет заголовком документа, но наименее важным из всех.

И обратите внимание, несмотря на то что отображение заголовков в браузере визуально отличает их, заголовки не следует использовать для стилизации документа. Эти элементы предназначены только для передачи структуры вашей HTML-страницы, не более того.

Стилизацию следует делать с помощью CSS, так как структура документа важна для поисковых машин. То, что помечено как h1, очевидно, является наиболее важным и обобщенным описанием содержания этой страницы. Для SEO крайне важно использовать тег h1, и он должен содержать формулировку, которая передает центральную тему остального содержания.


header element - Some header information goes here. Usually consists of company logo, some tag line, etc. Sometimes, navigation is contained in the header as well.
nav (short for navigation) element - Usually contains links to different parts of the web site.

Main Heading of the Page (hard not to have it)

Section 1
Article 1
Article 2
Article 3
Section 2
Article 4
Article 5
Article 6
Regular DIV element
ASIDE - Some information that relates to the main topic, i.e., related posts.
Copyright 2015

```
  <meta charset="utf-8">
  <title>Heading Elements</title>
</head>
<body>
  <header>
    header element - Some header information goes here. Usually consists of company logo, some tag line, etc.
    Sometimes, navigation is contained in the header as well.
    <nav>nav (short for navigation) element - Usually contains links to different parts of the web site.</nav>
  </header>
  <h1>Main Heading of the Page (hard not to have it)</h1>
  <section>
    Section 1
    <article>Article 1</article><article>Article 2</article><article>Article 3</article>
  </section>
  <section>
    Section 2
    <article>Article 4</article><article>Article 5</article><article>Article 6</article>
    <div>Regular DIV element</div>
  </section>
  <aside>
    ASIDE - Some information that relates to the main topic, i.e., related posts.
  </aside>
  <footer>
    Copyright 2015
  </footer>
</body>
</html>
```

В этом примере вы можете увидеть новый тег header, который содержит такую информацию о документе, как логотип компании, слоган, навигацию.

HTML тег <nav> содержит ссылки, ведущие на другие страницы сайта или на разделы текущей веб-страницы. Эти ссылки позволяют пользователю перемещаться по сайту.

Большинство сайтов имеют горизонтальное или вертикальное меню, располагающееся в верхней части страницы, это и является содержимым элемента <nav>.

В этом примере, у нас есть также элементы section. И внутри каждого тега section у нас есть элементы article.

Элемент section представляет общий раздел документа. И раздел section в этом контексте представляет собой тематическую группу контента, обычно с заголовком.

Элемент article представляет собой полную или автономную композицию в документе. Это может быть сообщение на форуме, статья, запись в блоге, пользовательский комментарий, интерактивный виджет или гаджет или любой другой независимый элемент контента.

Когда элементы article являются вложенными, внутренние элементы article представляют статьи, которые в принципе связаны с содержимым внешней статьи. Например, запись в блоге на сайте может представлять комментарии пользователей как вложенные элементы статьи.

Обычно статья помещается внутри элемента section. Тем не менее, это не всегда так, и, конечно же, нет жесткого правила по этому поводу. Вполне возможно, что в статье также могут быть разделы.

Далее в этом примере у нас есть тег aside.

Тег aside – это элемент, который сообщает, что внутри него есть что-то, что связано с основным содержимым страницы, но не в такой прямой связи, как основное содержимое.

И содержимое тега <aside> часто размещается в документе в виде боковой панели.

И, наконец, у нас есть тег <footer> нижнего колонтитула или подвала, который содержит некоторую информацию о сайте, такую как авторские права, контакты, карта сайта и т.д.

И в одном документе может быть несколько элементов <footer>.

Теперь следует отметить, что все эти теги являются элементами блочного уровня. Так что визуально мы могли бы просто везде использовать теги div. Однако, если вы посмотрите на код, станет очевидным, насколько легче его читать и понимать с этими специальными тегами и понимать структуру этой HTML-страницы, так как эти элементы передают некий смысл.

Теперь, давайте поговорим о HTML-списках.

```
<!DOCTYPE html>
<html>
<body>

<h1>The ul element</h1>

<ul>
  <li>Coffee</li>
  <li>Tea</li>
  <li>Milk</li>
</ul>

</body>
</html>
```

The ul element

- Coffee
- Tea
- Milk

```
<!DOCTYPE html>
<html>
<body>

<h1>The ol element</h1>

<ol>
  <li>Coffee</li>
  <li>Tea</li>
  <li>Milk</li>
</ol>
```

The ol element

1. Coffee
2. Tea
3. Milk

И списки – это невероятно полезная структура HTML, позволяющая группировать связанный контент. Если подумать, мы, как обычные люди, думаем о различных вещах в терми-

нах списков. Списки дел, списки покупок и так далее. И поэтому совершенно естественно, что списки есть и в HTML.

И в HTML есть два вида списков – упорядоченные списки и неупорядоченные.

Для того чтобы создать неупорядоченный список, нужно указать тег ul и заключить в него весь контент списка. При этом каждый элемент в списке находится в теге li.

При отображении в браузере такого списка, каждый его элемент помечается кружком.

Если мы скопируем и вставим этот HTML-код в валидатор W3C, он покажет, что наша HTML-страница действительна. Но давайте посмотрим, что произойдет, если мы возьмем один из элементов ненумерованного списка и просто удалим теги li вокруг него.

Если мы проверим этот код снова в валидаторе W3C, мы увидим, что HTML код теперь недействителен, и причина в том, что внутри тега ul не разрешен текст.

Единственное, что разрешено внутри элемента ul, – это элемент li. Все остальное не допускается.

Для создания упорядоченного списка тег ul заменяется тегом ol, но элементы li при этом точно такие же.

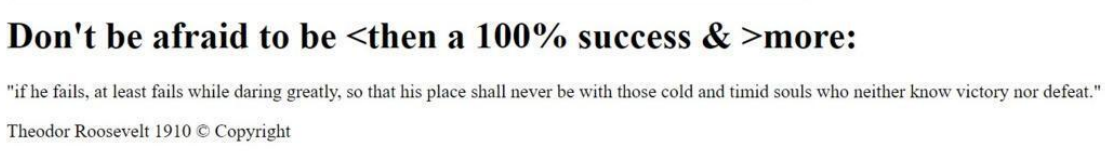
В браузере элементы упорядоченного списка помечаются порядковым номером.

Теперь, что касается контента, так как HTML использует определенные символы для своего синтаксиса, нам нужен способ различать эти символы как HTML код и те же символы как содержимое.

Если мы хотим, чтобы браузер интерпретировал специальные символы HTML как обычный контент, нам нужен способ указать браузеру не интерпретировать их как HTML код.

В частности, есть три символа, которые всегда следует экранировать, чтобы они не вызывали проблем с рендерингом.

```
<!DOCTYPE html>
<html>
<head>
  <meta charset="utf-8">
  <title>HTML Entities</title>
</head>
<body>
<h1>Don't be afraid to be &lt;then a 100% success &amp; &gt;more:</h1>
<p>
  &quot;if he fails, at least fails while daring greatly,
  so that his place shall never be with those cold and timid souls
  who neither know victory nor defeat.&quot;
</p>
<p>Theodor Roosevelt 1910 &copy; Copyright</p>
</body>
</html>
```



Это символы <, > и &. И эти символы зарезервированы в HTML. Например, если вы используете в тексте знаки меньше (<) или больше (>), браузер может перепутать их с тегами.

И для отображения зарезервированных символов в HTML используются сущности символов.

Вместо использования символа меньше <, вы должны использовать объект HTML, который начинается с амперсанда &, а затем следует lt;. и браузер интерпретирует это как символ меньше <.

Аналогично, для символа больше > нужно использовать >.
А для символа амперсанда & нужно использовать &.

Result	Description	Entity Name	Entity Number
	non-breaking space		
<	less than	<	<
>	greater than	>	>
&	ampersand	&	&
"	double quotation mark	"	"
'	single quotation mark (apostrophe)	'	'
¢	cent	¢	¢
£	pound	£	£
¥	yen	¥	¥
€	euro	€	€
©	copyright	©	©
®	registered trademark	®	®

На самом деле HTML содержит целую массу сущностей символов HTML.

И одним из наиболее распространенных является символ авторского права. И основная причина, по которой мы используем для него сущность символа, заключается в том, что этот символ не очень-то и легко найти на любой клавиатуре.

Поэтому мы можем использовать вместо символа авторского права – ©.

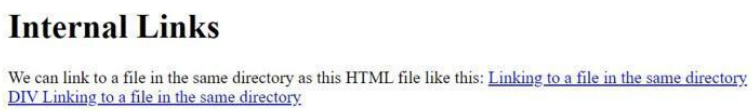
Еще одна полезная сущность HTML, которая очень часто используется, это , что означает неразрывный пробел. При этом слова с неразрывным пробелом всегда будут на одной строке, то есть браузером эти слова будут переноситься на новую строку вместе.

Когда мы сожмем браузер и сделаем его чуть менее широким, эти слова либо вместе переместятся на следующую строку, либо останутся на одной строке, но они не будут разделены.

Теперь, давайте поговорим о ссылках на другие документы и способы их создания на HTML-странице.

```html
<!doctype html>
<html>
<head>
  <meta charset="utf-8">
  <title>Internal Links</title>
</head>
<body>
  <h1>Internal Links</h1>
  <section>
    We can link to a file in the same directory as this HTML file like this:
    <a href="same-directory.html" title="same dir link">Linking to a file in the same directory</a>

    <a href="same-directory.html" title="same dir link">
      <div> DIV Linking to a file in the same directory</div>
    </a>
  </section>
</body>
</html>
```



Итак, первый тип ссылок, который мы собираемся рассмотреть, – это внутренние ссылки.

И мы создаем ссылки с помощью элемента <a> с атрибутом href.

Атрибут href указывает гипертекстовую ссылку. И значение href может быть как относительным, так и абсолютным URL-адресом.

В нашем случае, так как мы обсуждаем внутренние ссылки, которые указывают на внутренние веб-страницы сайта, ссылки являются относительными URL-адресами.

Также полезно указывать атрибут title для тега a. Атрибут title используется программами чтения с экрана, которые помогают слабовидящим людям просматривать веб-страницу.

Далее, содержимое между открывающим и закрывающим тегами элемента <a> – это содержимое, по которому вы сможете щелкнуть, чтобы перейти к ссылке href.

Теперь, обратите внимание на второй пример тега a. В этом примере мы окружаем тег div тегом a. Другими словами, этот тег div будет контентом ссылки, по которому можно будет щелкнуть.

Таким образом, первая ссылка является встроенным тегом, так как она не инициирует переход на новую строку. Но во втором случае, мы окружаем блочный тег div тегом a.

А тег a в HTML5 одновременно является и встроенным элементом, и элементом уровня блока. Именно это позволяет нам взять тег a и вложить в него тег div, контент которого начинается с новой строки.

Авторы спецификации HTML5 знали, что во многих случаях нужно иметь возможность щелкнуть по целой области HTML документа, например по логотипу компании, чтобы перейти на ее сайт.

До HTML5 приходилось использовать всевозможные приемы, чтобы добиться такого же эффекта, потому что тег a был только встроенным тегом, и мы не могли обернуть тег a вокруг тега div.

```html
<!doctype html>
<html>
<head>
  <meta charset="utf-8">
  <title>Links</title>
</head>
<body>
  <h1 id="top">External Links</h1>
  <section>
    <p>
      Let's link to a Facebook page!
      <!-- Link to Facebook page WITH TARGET-->
      <a href="http://www.facebook.com/"
      target="_blank" title="Like Our Page!">Facebook Page</a>
    </p>
  </section>
</body>
</html>
```

Теперь, о внешних ссылках.

На самом деле во внешних ссылках нет ничего особенного, кроме того, что их значение href обычно начинается с http://, потому что обычно внешние ссылки указывают на документы других веб-сайтов.

Однако здесь есть одна особенность элемента a, которая довольно часто используется в сочетании с внешними ссылками. И это целевой атрибут target.

Когда для атрибута target установлено значение _blank, это заставляет браузер открывать страницу по ссылке в новой вкладке или в новом окне.

Таким образом сайт не закрывается, а новая страница открывается в новой вкладке, так что вернуться на сайт после просмотра страницы по ссылке будет удобно.

```html
<!doctype html>
<html>
<head>
  <meta charset="utf-8">
  <title>Links</title>
</head>
<body>
  <h1 id="top">Links to Sections of The Same Page</h1>
  <section>
    <ul>
      <!-- Link to every section in the page -->
      <li><a href="#section1">#section1</a></li>
      <li><a href="#section2">#section2</a></li>
    </ul>
  </section>
  <section id="section1">
    <h3>(#section1) Section 1</h3>
    <p>Lorem ipsum dolor sit amet</p>
  </section>
  <section id="section2">
    <h3>(#section2) Section 2</h3>
    <p>Lorem ipsum dolor sit amet</p>
    <p>Back to top: <a href="#top">Back to Top</a></p>
  </section>
</body>
</html>
```

Следующий тип ссылки – это ссылка на фрагмент страницы. Эта ссылка позволяет навигацию по большому документу, так что не нужно прокручивать его и искать нужный вам фрагмент документа.

Такая ссылка имеет специфический формат атрибута href. Здесь есть символ #, за которым следует идентификатор фрагмента, как в этом примере section1, section2.

Таким образом эта ссылка указывает на раздел страницы внутри элемента, который имеет идентификатор с атрибутом id.

И вы можете использовать любой тег с идентификатором раздела.

Обратите внимание, что имя раздела не содержит знака #. Только ссылка на этот раздел содержит знак #.

И здесь у нас даже есть ссылка «назад к началу», которая указывает на начало документа – тег h1 с идентификатором top.

И если вы щелкнете по такой ссылке, идентификатор фрагмента появится в URL-адресе, например index.html#section1, и вы можете скопировать и использовать этот URL-адрес в качестве другой ссылки. Так что при открытии страницы можно будет сразу перейти к разделу, на который указывает идентификатор.

Теперь, давайте поговорим о том, как включать изображения в HTML-документы.

```
<!DOCTYPE html>
<html>
<head>
  <meta charset="utf-8">
  <title>Displaying Images</title>
</head>
<body>
<h1>Don't be afraid to be &lt;than a 100% success:</h1>
<!-- изображение отображается прямо перед цитатой -->
<p>
  <img src="picture-with-quote.jpg" width="400" height="235" alt="Picture with a quote">
  &quot;who at the worst, if he fails, at least fails while daring greatly,
  so that his place shall never be with those cold and timid souls who neither know victory nor defeat.&quot;
</p>
<p>
<img src="https://picsum.photos/536/354" >
</p>
<p>Theodore Roosevelt 1910 &copy; Copyright</p>
</body>
</html>
```

В этом примере первое изображение отображается прямо перед цитатой.

И здесь у нас есть соответствующий комментарий. Так выглядит HTML-комментарий.

Комментарий в HTML начинается с угловой скобки, восклицательного знака, далее тире, тире, а затем комментарий закрывается тире, тире, и угловой скобкой.

И браузер полностью проигнорирует этот комментарий и не отобразит его.

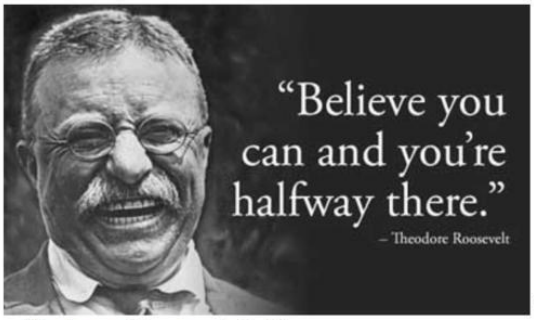
Теперь давайте посмотрим на тег изображения <img>.

Изображение отображается в HTML с помощью этого тега img, который является сокращением от слова image. А атрибут src тега img – это URL-адрес, указывающий на файл изображения.

И вы можете заметить, что этот URL-адрес ничем не отличается от href, который вы видели у тега ссылки <a>. Это может быть относительный URL-адрес или абсолютный URL-адрес, если он содержит внешнюю ссылку.

Далее мы указываем ширину и высоту изображения. Хотя это и не обязательно, рекомендуется всегда указывать ширину и высоту. И еще есть атрибут alt, который используется программами чтения с экрана, помогающими людям с нарушениями зрения.

Don't be afraid to be <than a 100% success:



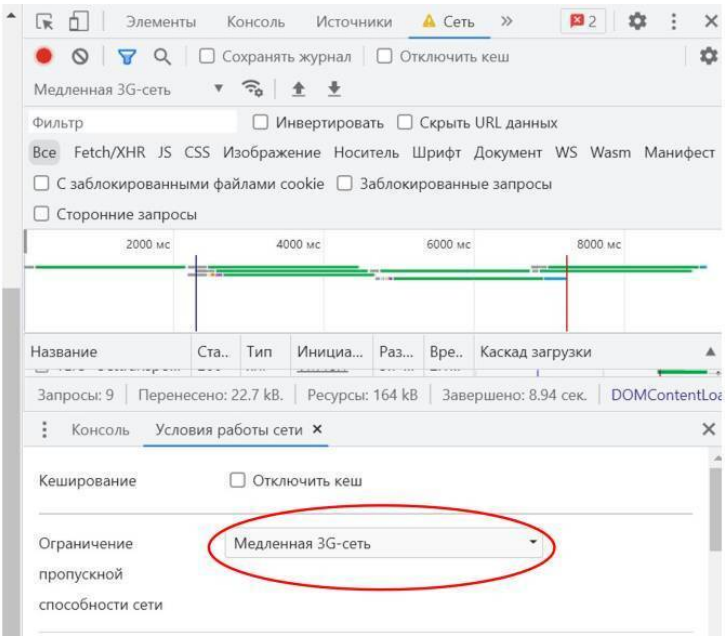
"who at the worst, if he fails, at least fails while daring greatly, neither know victory nor defeat."

Как вы видите, это изображение отображается в браузере в одной строке с текстом, так как тег img является встроенным элементом. Если бы это было не так, изображение было бы на своей собственной строке.

Теперь давайте посмотрим на другой тег img. Здесь используется внешняя ссылка, указывающая на изображение в Интернете.

И здесь не указана высота и ширина изображения. Поэтому оно будет отображаться со своими изначальными размерами. И если эти размеры очень большие, изображение может занять всю вашу HTML страницу, не оставив места для остального контента. Вот почему рекомендуется указывать высоту и ширину изображения.

И еще, если для загрузки изображения требуется время, страница может дергаться во время загрузки. Но если вы используете ширину и высоту изображения, вы говорите браузеру, чтобы он зарезервировал пространство для изображения и загружал его в эту область. Тогда страница не будет дергаться.



Вы можете моделировать скорость загрузки в инструментах разработчика браузера. Здесь вы можете установить ограничение сети и посмотреть, как выглядит страница во время загрузки изображения.

Еще хуже, если вы полагаетесь на изображение, чтобы создать какой-то визуальный макет для своей страницы, и по какой-то причине изображение не загрузилось, например, URL-адрес может быть неверным. Если мы полагаемся на определенное пространство между цитатой и нижним колонтитулом страницы, мы больше не сможем видеть это пространство, так как изображение не загружается.

Однако, если мы укажем ширину и высоту изображения, даже несмотря на то, что изображение не загружается, визуальный интервал и визуальный макет остаются правильными.

Вопросы

Вопрос 1

Основная цель HTML состоит в том, чтобы

(подсказка: что-то, что не сможет выполнить обычный текстовый файл с содержимым)

Отобразить содержимое веб-страницы для пользователя

Сообщить структуру контента

Сообщить браузеру, как расположить и выровнять содержимое в окне браузера.

Сообщить браузеру, что должно произойти после загрузки страницы

Вопрос 2

W3C (Консорциум World Wide Web) – единственная организация, которая диктует, как браузеры должны реализовывать HTML5.

Истина

Ложь

Вопрос 3

ВСЕ теги HTML5 должны иметь

Наличие открывающего тега

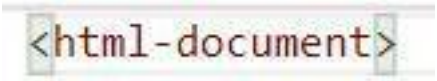
Наличие закрывающего тега

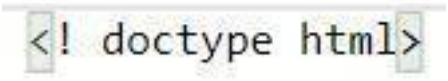
Иметь хотя бы 1 атрибут

Быть в нижнем регистре, т.е. <p>

Вопрос 4

Определите правильное объявление страницы HTML5





Вопрос 5

Что произойдет, если вы не укажете объявление HTML5?

Ничего. HTML5 очень либерально относится к своему объявлению, и это всего лишь рекомендация по его использованию.

Браузер автоматически вернется к предыдущей версии HTML.

Страница будет интерпретироваться в режиме причуд.

Страница вообще не отобразится.

Вопрос 6

Браузеры ВСЕГДА интерпретируют HTML последовательно, сверху вниз.

Истина

Ложь

Вопрос 7

Определите недопустимый фрагмент кода HTML ниже

1. Hey there!
I am just a lonely quiz answer. Will you click me? I am bored!

2. <section>
The sale numbers are in…
</section> You ARE a closer, Johnson!

3. <article>
<h2>Wow!
</h2>
You're ignoring all the other answers just to
look at me?! I AM special!
…
Wait! Where are you going?
What if I AM the answer you've been searching for???!

</article>

Вопрос 8

Метатеги передают информацию о браузере на сервер.

Истина

Ложь

Вопрос 9

Без применения каких-либо дополнительных стилей следующий фрагмент кода будет отображаться в браузере как какое количество строк текста? (Предположим, что браузер растянут достаточно широко, чтобы ни одна строка не переносилась).

Dear all,
I took this really cool book

 I think it's
my favorite book I've EVER taken!
Here is the URL for it:

about HTML, CSS and JS

Does anyone know how I can give this book 6
out of 5 stars?

Thank you,
–Yaakov.... !

Вопрос 10

Используя только HTML, как бы вы удостоверились, что 3 слова в HTML-документе ВСЕГДА появляются вместе в 1 строке, даже если текст переносится по словам, потому что окно браузера слишком узкое для этой текстовой строки?

Невозможно выполнить только с помощью HTML!

Поместите &nowrap; перед 1-м словом и после 3-го слова

Поместите после 1-го слова и после 2-го слова (без пробелов)

Поместите перед 1-м словом и после 3-го слова

Вопрос 11

Как заставить браузер открывать ссылку в новом окне или вкладке?

Укажите несколько специальных метатегов как часть страницы

Попросите пользователя щелкнуть ссылку правой кнопкой мыши и выбрать «Открыть в новой вкладке».

Включить атрибут target='_blank' как часть тега <a>

Включите атрибут target="new" как часть тега <a>

Вопрос 12

Несмотря на то, что атрибуты ширины и высоты тега img не требуются, всегда полезно их использовать.

Истина

Ложь

Введение в CSS

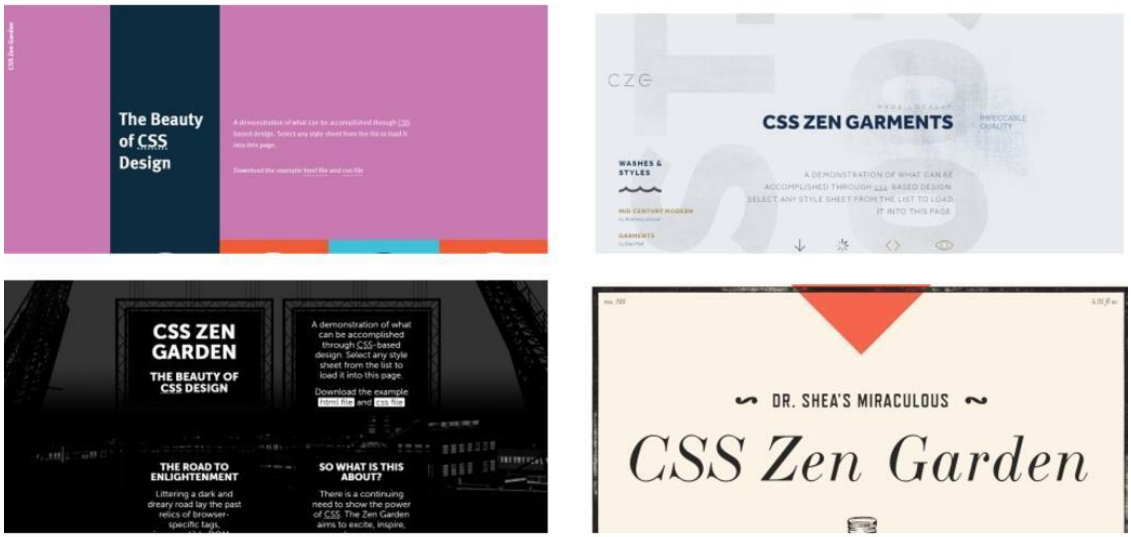
Мы уже говорили о том, что в Интернете главное – контент. И мы говорили о том, что HTML определяет структуру этого контента.

Но, хотя контент и является основой, очень важно и то, как пользователи воспринимают этот контент. И когда дело доходит до реального веб-сайта, одной структуры недостаточно, вы должны оформить свой контент таким образом, чтобы он был приятным и полезным для пользователя.

И использование цвета, позиционирования, размера и прочего является частью этого оформления.

Каскадные таблицы стилей, или CSS, – это технология, обеспечивающая возможность стилизации и оформления контента в Интернете.

http://csszengarden.com/



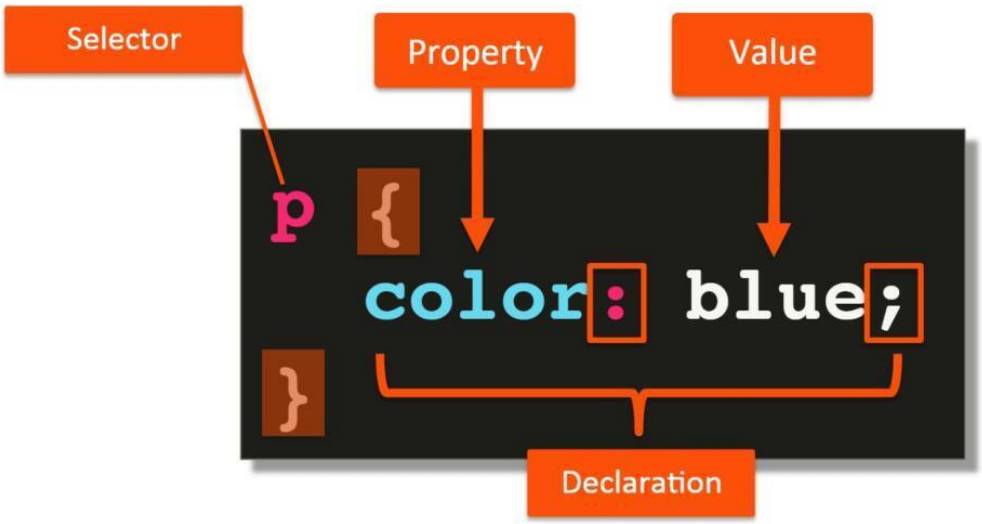
В качестве примера можно посмотреть сайт csszengarden.com, где один и тот же HTML файл с помощью CSS выглядит фантастически по-разному.

CSS или каскадные таблицы стилей работают следующим образом, они связывают определенные правила с элементами HTML.

```
<!DOCTYPE html>
<html>
<head>
<style>
body {
 background-color: lightblue;
}
h1 {
 color: white;
 text-align: center;
}
p {
 font-family: verdana;
 font-size: 20px;
}
</style>
</head>
<body>
<h1>My First CSS Example</h1>
<p>This is a paragraph.</p>
</body>
</html>
```

Эти правила управляют тем, как должно отображаться содержимое указанных элементов.

И хотя стилизация всей веб-страницы может быть довольно сложным процессом, определить простое правило CSS довольно просто.



Правило CSS состоит из селектора, в данном случае это p, тэг абзаца, и это говорит о том, что это правило должно применяться к содержимому каждого тэга абзаца на всей HTML-странице.

За селектором следуют открывающие и закрывающие фигурные скобки. И внутри этих фигурных скобок у нас есть объявление CSS, которое состоит из двух частей, свойства и значения.

Имя свойства предопределено спецификацией CSS, и для каждого свойства существует определенное количество предопределенных значений.

Каждое свойство отделяется от значения двоеточием и заканчивается точкой с запятой.

С технической точки зрения, точка с запятой не является обязательным требованием, но это лучшая практика, и вы должны всегда ее использовать.

```
body {
  background-color: lightblue;
}

h1 {
  color: white;
  text-align: center;
}

p {
  font-family: verdana;
  font-size: 20px;
}
```

Каждое правило CSS может состоять из нескольких объявлений. Например, в этом случае мы указываем, что содержимое тегов абзаца на нашей HTML-странице должно иметь определенный шрифт и определенный размер шрифта.

Теперь, набор этих правил CSS называется таблицей стилей.

```
<head><style>
p {
 color: blue;
 font-size: 20px;
 width: 200px;
}
h1 {
 color: green;
 font-size: 36px;
 text-align: center;
}
</style></head>
<body>
<h1>Anatomy of a CSS Rule</h1>
<h2>Subheading 1</h2>
<p>CSS works by associating rules with HTML elements</p>
<h2>Subheading 2</h2>
<p>Lorem ipsum dolor sitamet, consectetur adipisicing elit.</p>
</body>
</html>
```

В этом HTML-документе у нас есть заголовок h1, пара заголовков h2 и пара абзацев между ними, обозначенные тегом абзаца p. И что мы хотим сделать, так это придать этим элементам индивидуальный стиль. И мы разместим таблицу стилей прямо в разделе заголовка нашего HTML-документа.

И в этой таблице мы видим тег абзаца. И мы изменим его цвет на синий, размер шрифта на 20 пикселей и ширину на 200 пикселей, что означает, что он будет занимать область экрана шириной 200 пикселей.

Далее для тега h1, мы определяем зеленый цвет и увеличиваем его размер шрифта, и мы также выравниваем его по середине экрана.

Так эта страница выглядит в браузере.

И если вы посмотрите на подзаголовок 1, а затем на подзаголовок 2, вы увидите, что они выделены жирным шрифтом и немного больше, чем обычный текст.

Откуда взялся этот стиль? Ведь мы не определяли стиль для тега h2.

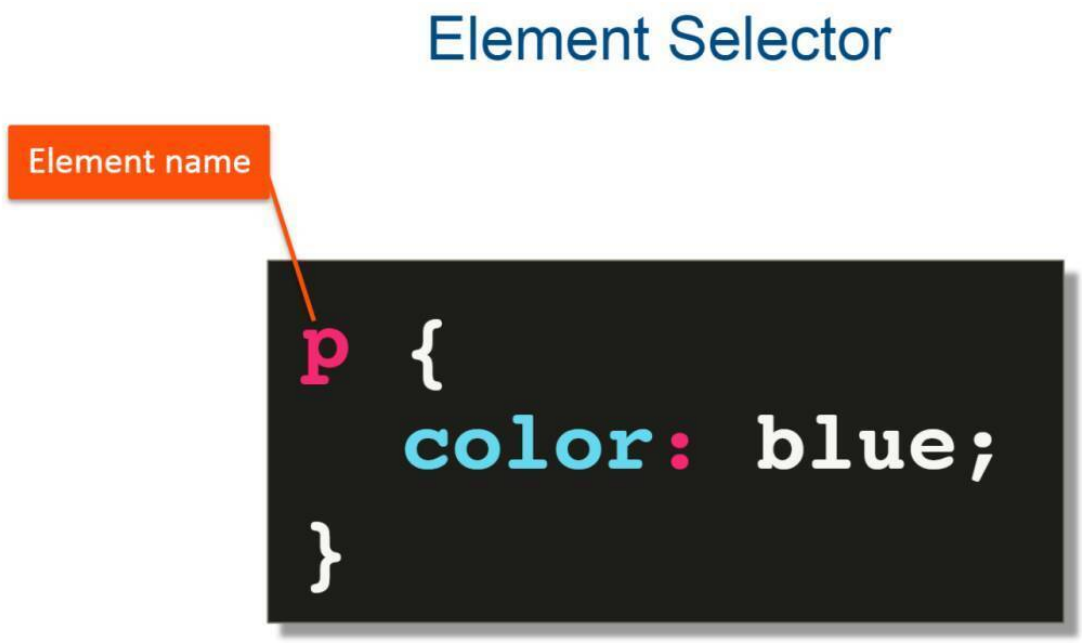
И этот стиль исходит от самого браузера. Каждый браузер поставляется с определенными стилями по умолчанию, которые применяются к различным элементам HTML.

Таким образом, во многих случаях наша работа заключается в том, чтобы переопределить эти стили браузеров по умолчанию.

Как мы уже говорили, правило CSS состоит из селектора, за которым следуют открывающие и закрывающие фигурные скобки с объявлениями CSS, состоящими из двух частей, свойства и значения.

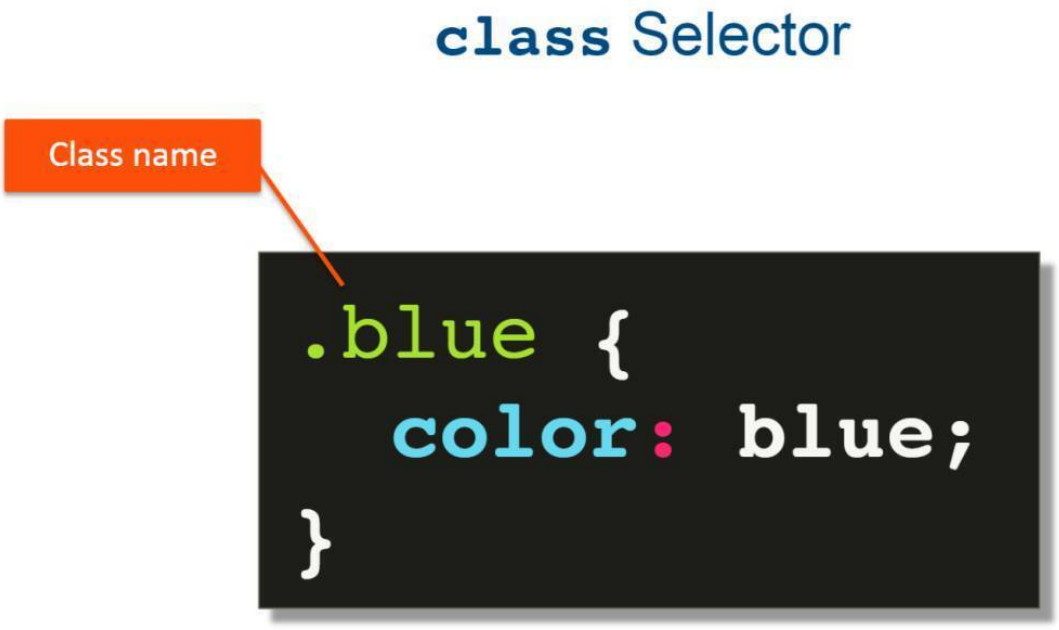
И селекторы CSS используются для определения того, к какому элементу HTML или набору элементов следует применить объявления CSS. Браузер использует свой API-интерфейс для обхода документа и выбора элементов, соответствующих данному селектору.

И существует три разных типов селекторов: элемент, класс и идентификатор.



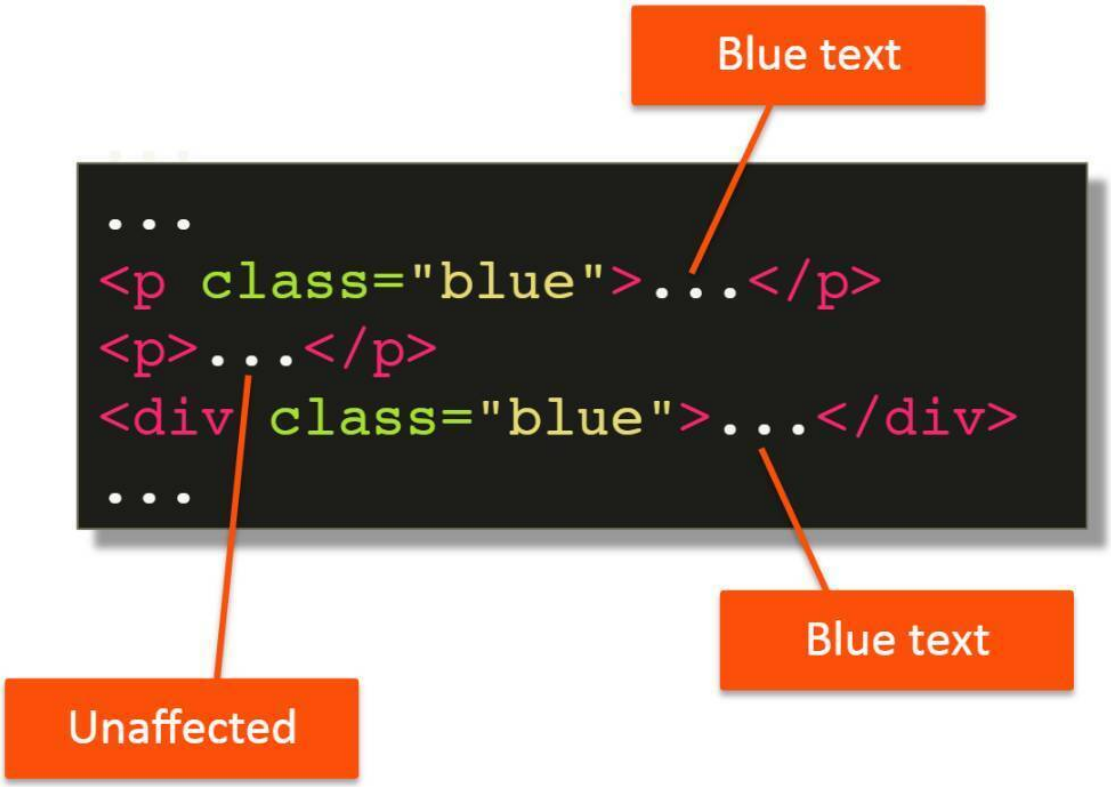
Первый тип селектора – элемент – это просто указание имени элемента. Например, в этом случае селектор p говорит о том, что текст каждого абзаца в нашем HTML-документе должен быть синего цвета.

И это объявление никак не влияет на другие элементы, содержащие текст. Например, у нас может быть элемент div, содержащий текст, но на этот текст не повлияет наше правило CSS для абзаца.



Далее идет селектор класса, который указывается точкой и именем класса. В этом случае мы создаем класс blue CSS, который будет окрашивать синим цветом.

И селектор класса требует изменения вашего HTML-документа, так как каждый элемент, к которому вы хотите применить этот класс, должен иметь атрибут класса с именем этого класса.

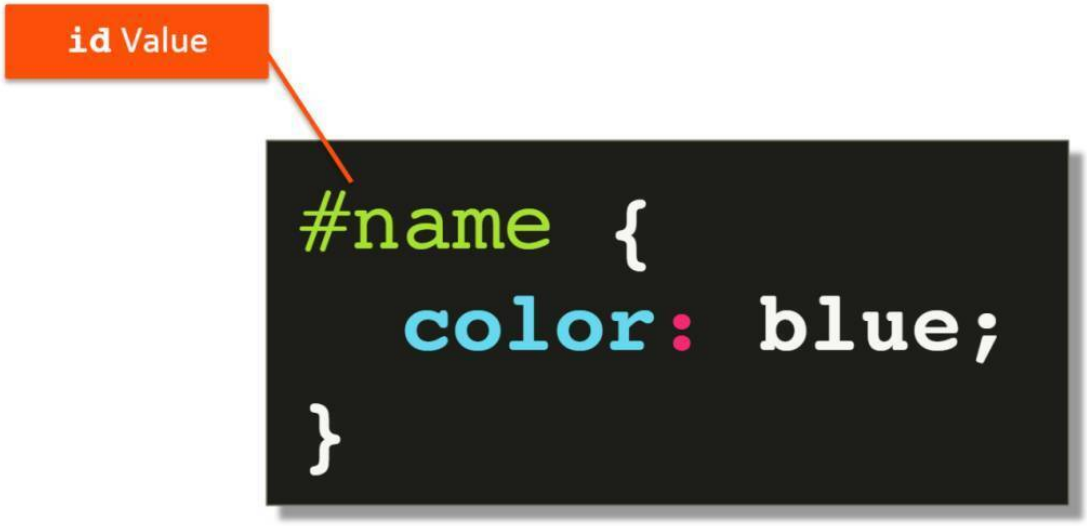


В этом случае у нас есть p, тег абзаца и тег div, и оба имеют атрибут class="blue" и, следовательно, их текстовое содержимое будет окрашено в синий цвет.

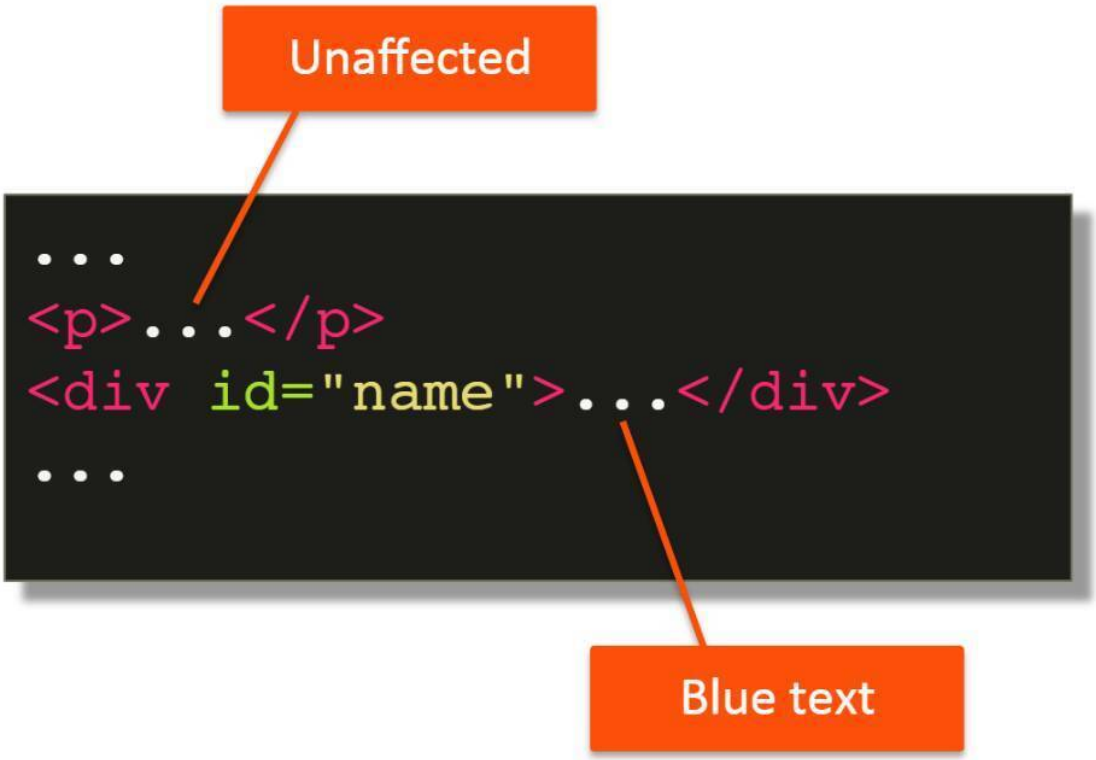
Обратите внимание, что это совершенно не влияет на другой абзац, не помеченный атрибутом class="blue".

И обратите внимание на разницу между тем, как вы определяете класс, и тем, как вы используете класс. Вы всегда определяете класс с точкой перед именем. И между точкой и именем класса не должно быть пробела. Однако, когда вы используете класс, вы не используете точку в его имени, вы просто используете его имя.

И последний тип селектора – это селектор id.



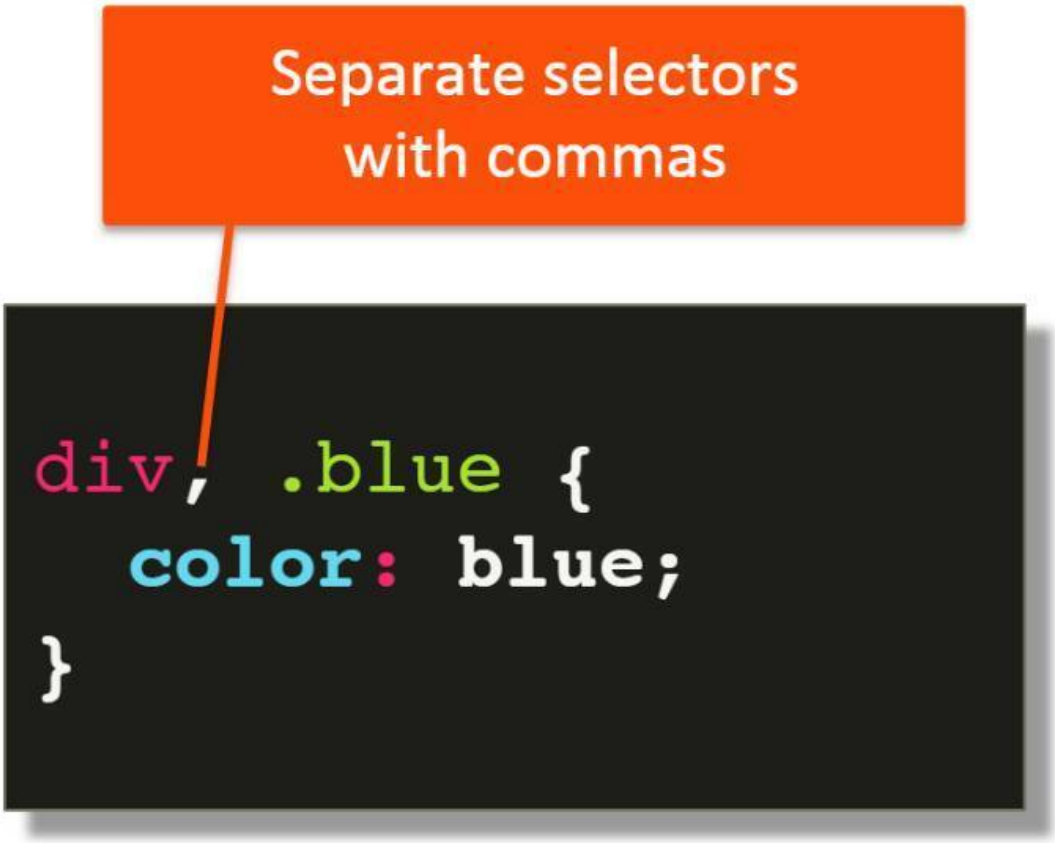
Вы указываете селектор идентификатора с помощью значения идентификатора элемента в вашем HTML-документе, которому предшествует знак решетки.



Так, например, если у вас есть элемент div с id="name", а затем вы указываете правило CSS с помощью селектора #name, объявления CSS будут применяться к содержимому элемента div с его значением id name.

Селектор идентификатора определяется знаком решетки, за которым следует значение идентификатора в вашем HTML-документе. И опять же, у вас не может быть никакого промежутка между ними.

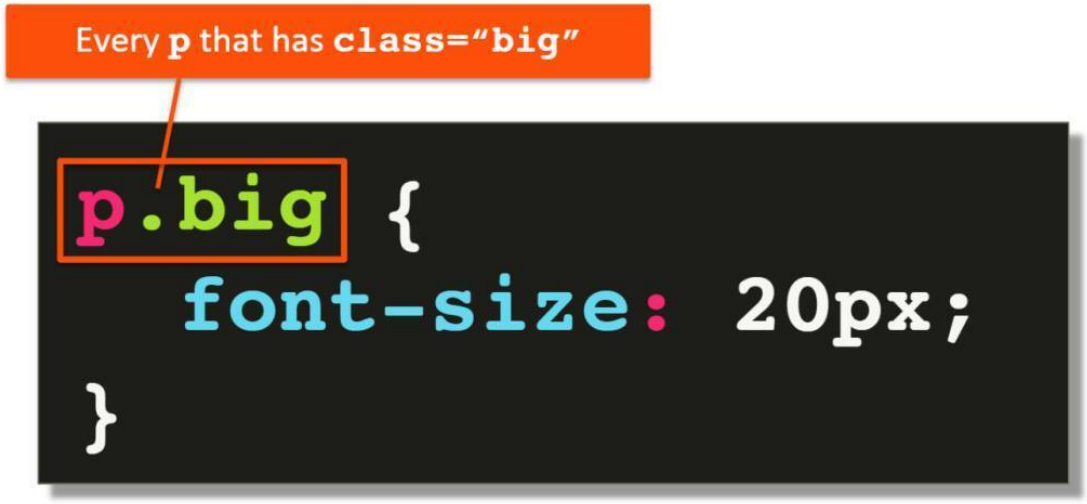
Теперь, чтобы писать более эффективные правила, CSS позволяет нам сгруппировать несколько селекторов в одно правило CSS.



Здесь у нас есть два селектора, сгруппированные вместе, div и также селектор класса blue, так как они оба используют одно и то же объявление CSS.

Помимо группировки селекторов существует объединение или комбинирование селекторов. И объединение селекторов – это очень мощная техника, позволяющая более точно выделять элементы документа.

Первый способ комбинирования селекторов – это селектор элемента с селектором класса.

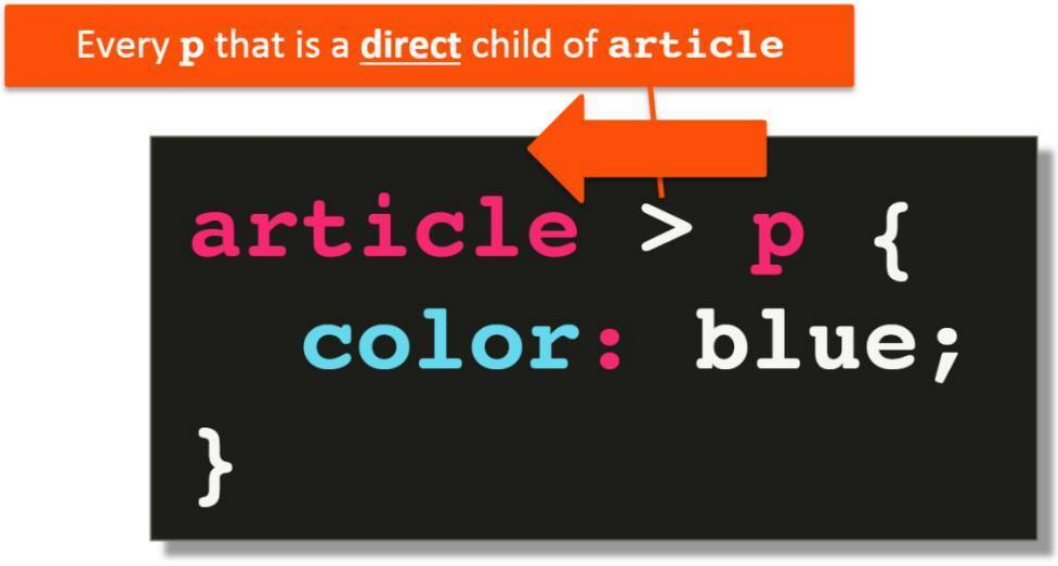


Здесь у нас есть селектор элемента p, за которым сразу без пробелов следует точка, которая является селектором класса. И это говорит о том, что мы хотим выделить каждый элемент p, который имеет атрибут класса со значением big.

И обратите внимание на отсутствие пробела между этими селекторами. Если вы поставите пробел, это будет означать совершенно другую комбинацию.

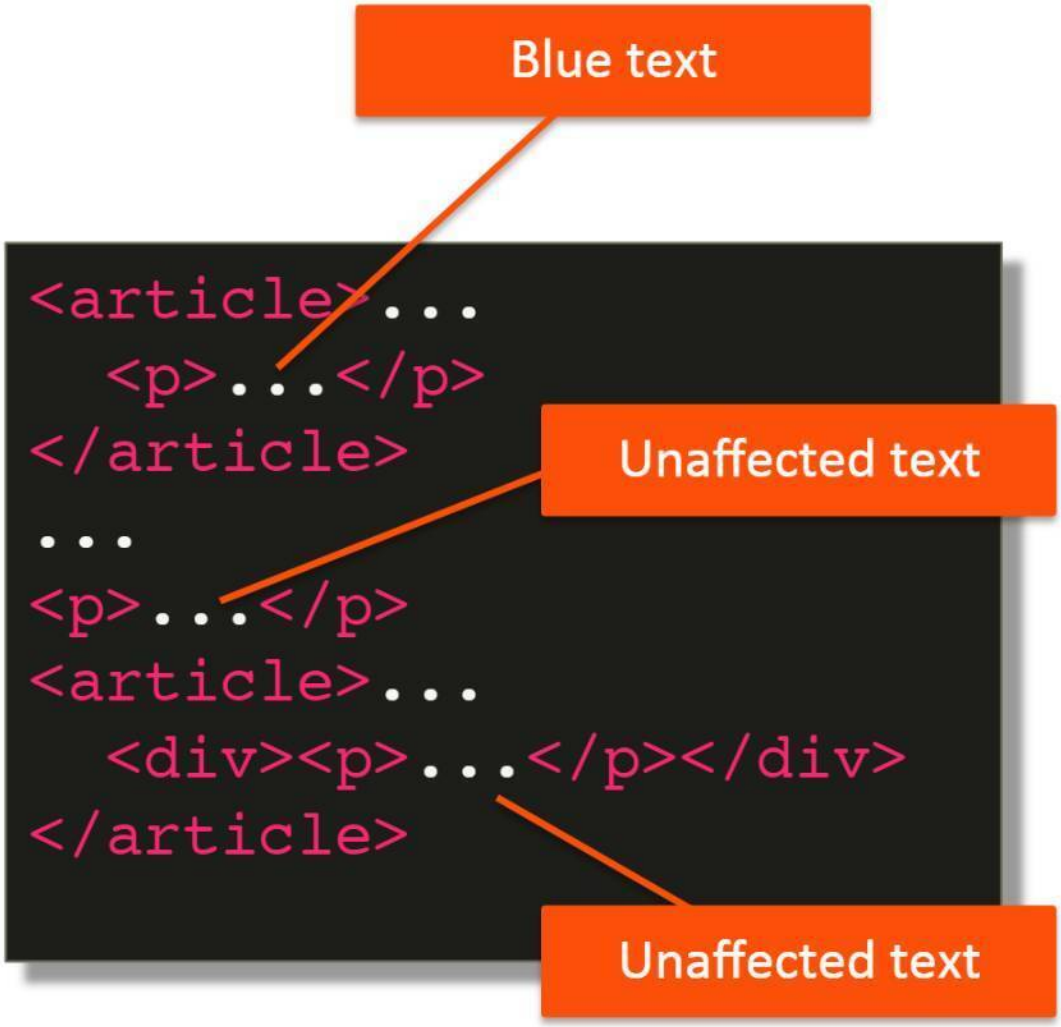
Этот метод довольно часто используется, когда у вас есть оправило CSS, которое применяется к различным элементам, но вы хотите, чтобы для конкретного элемента это правило изменилось.

Следующим типом комбинации селекторов является дочерний селектор.

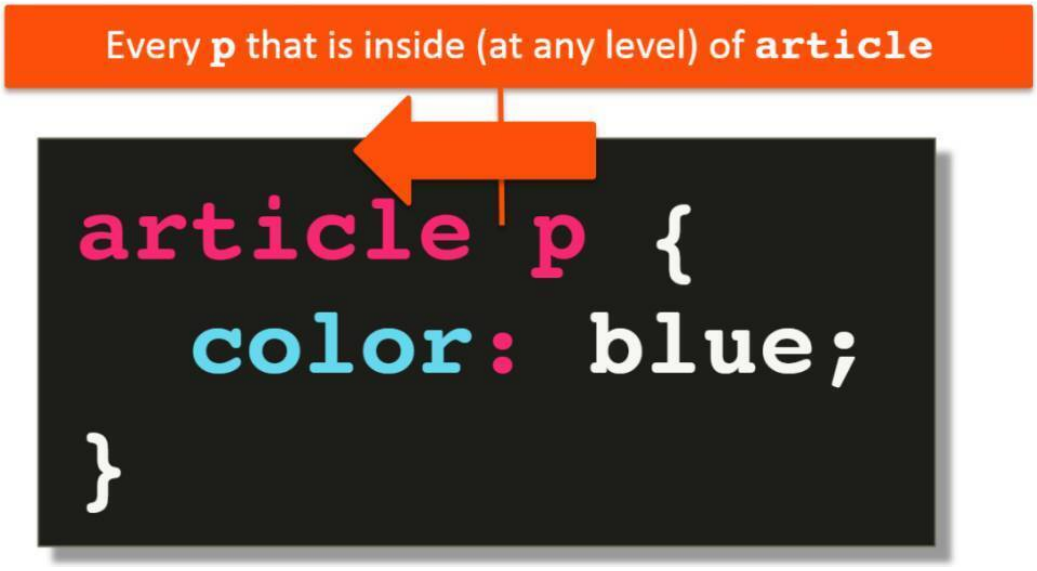


В этом случае указывается селектор, за которым следует угловая скобка, за которой следует еще один селектор. И вы читаете эту комбинацию справа налево.

И в этом случае мы выделяем каждый элемент p, который является прямым дочерним элементом элемента article. Таким образом, в этом примере элемент p, который является прямым дочерним элементом элемента article, получит стиль синего цвета, в отличие от обычного элемента p.

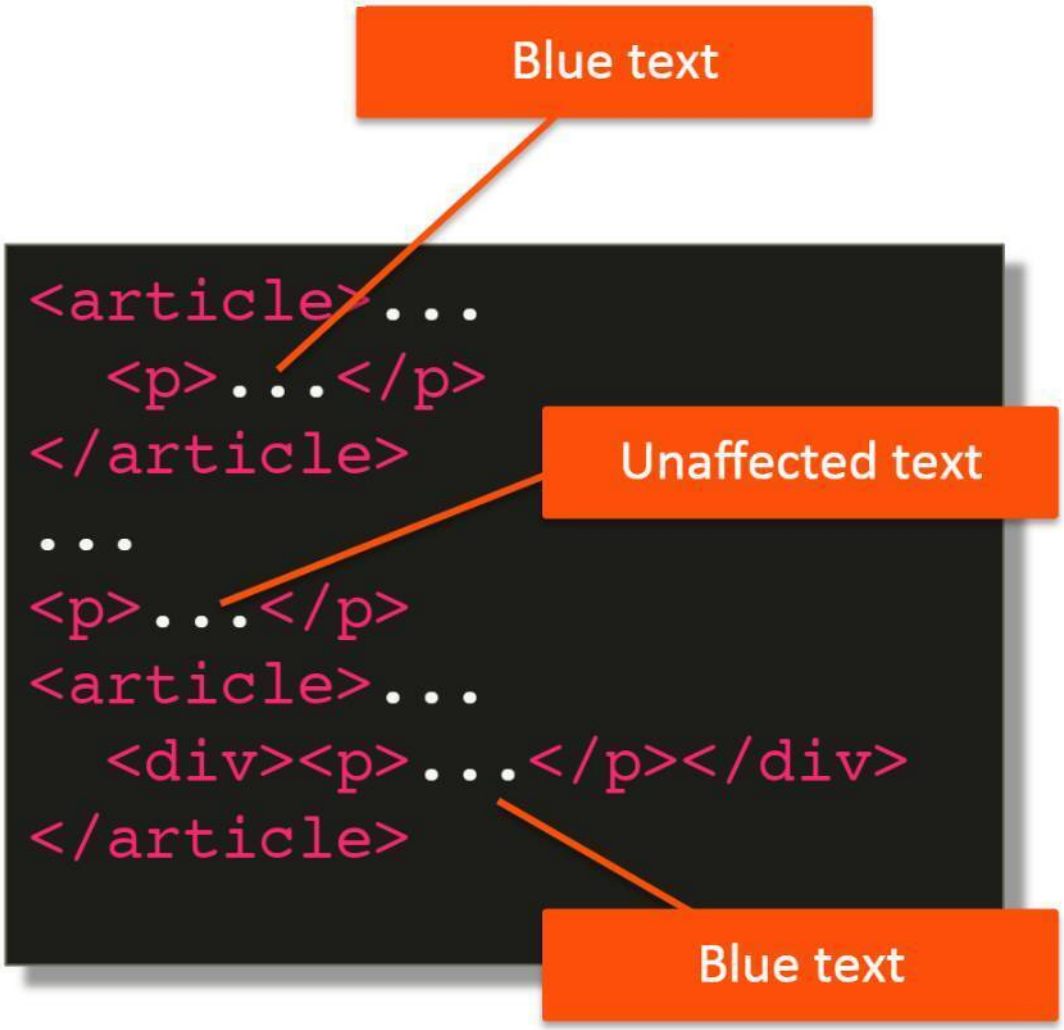


Другой тип комбинации селекторов – это селектор потомков.



И синтаксис этой комбинации – селектор пробел родительский селектор. И так же, вы читаете эту комбинацию справа налево.

В этом случае мы выделяем каждый элемент p, который находится внутри элемента article на любом уровне, что означает, что даже если это не прямой дочерний элемент, даже если он глубоко внутри, до тех пор, пока один из его родителей является элементом article, это правило будет применяться.



Теперь, нужно отметить, что эти комбинации селекторов не ограничиваются селекторами элементов. Так, например, у нас может быть комбинация .colored p элемента с классом, или комбинация класса с классом.

Помимо простых селекторов (выбор элементов на основе тега, идентификатора, класса), есть также селекторы псевдоклассов, которые выбирают элементы на основе определенного их состояния.

Селекторы псевдоклассов дают возможность стилизации документа на основе взаимодействия пользователя со страницей. Например, мы можем сделать так, чтобы стиль элемента менялся, когда пользователь наводит курсор мыши на элемент.

Селектор псевдокласса определяется с помощью указания некоторого селектора с двоеточием и предопределенным именем псевдокласса. И существует множество селекторов псевдоклассов.

```
a:link {
  background-color: yellow;
}
```

```
<!DOCTYPE html>
<html>
<head>
<style>
a:link {
  background-color: yellow;
}
</style>
</head>
<body>

<h1>Demo of the :link selector</h1>

<p>The :link selector style links to pages you have not visited yet:</p>

<a href="https://www.w3schools.com">W3Sschools</a>
<a href="http://www.wikipedia.org">Wikipedia</a>

</body>
</html>
```

Demo of the :link selector

The :link selector style links to pages you have not visited yet:

W3Sschools Wikipedia

Например, селектор ссылок «:link» стилизует не посещенные ссылки. Здесь мы окрашиваем не посещенную ссылку в желтый цвет.

```
a:visited {
  background-color: pink;
}
```

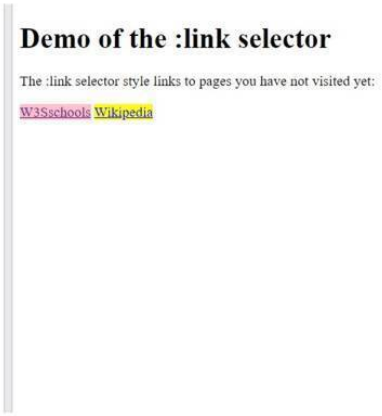
```
<!DOCTYPE html>
<html>
<head>
<style>
a:link {
  background-color: yellow;
}
a:visited {
  background-color: pink;
}
</style>
</head>
<body>

<h1>Demo of the :link selector</h1>

<p>The :link selector style links to pages you have not visited yet:</p>

<a href="https://www.w3schools.com">W3Sschools</a>
<a href="http://www.wikipedia.org">Wikipedia</a>

</body>
</html>
```



Также мы можем стилизовать посещенную ссылку с помощью селектора «:visited».

```
a:hover {
  background-color: yellow;
}
```

Селектор «:hover» используется для выбора и стилизации элементов при наведении на них курсора мыши. И селектор «:hover» можно использовать для любых элементов, а не только для ссылок или кнопок.

При использовании селектора «:hover» для ссылок, он должен идти после селекторов «:link» и «:visited» в определении CSS.

```
a:active {
  background-color: yellow;
}
```

Селектор «:active» используется для выбора и стилизации активной ссылки, т.е. ссылки при нажатии на нее. Хотя селектор «:active» можно использовать для всех элементов, а не только для ссылок.

При использовании селектора «:active», он должен идти после селектора «:hover» в определении CSS, что соответствует логики действий пользователя.

```
<style>
/* Selects the second element of div siblings */
div:nth-child(2) {
  background: red;
}

/* Selects the second li element in a list */
li:nth-child(2) {
  background: lightgreen;
}

/* Selects every third element among any group of siblings */
:nth-child(3) {
  background: yellow;
}
</style>
```

Селектор «:nth-child(n)» выбирает каждый элемент, который является n-м дочерним элементом своего родителя. При этом n может быть числом, ключевым словом (четный или нечетный, odd или even) или формулой, например (an + b), где a представляет размер цикла, n – счетчик (начиная с 0), а b – значение смещения.

В этом примере, элемент «This is some text.» в первом теге div никак не стилизован, далее такой же элемент во втором теге div выделен красным цветом. В списке содержимое второго элемента выделено зеленым цветом, и содержимое каждого третьего элемента выделено желтым цветом.

Теперь, ваш выбор размещения стилей в одном или другом месте не только влияет на возможность повторного использования стилей, но также влияет на то, какие объявления стилей переопределяют другие объявления стилей.

```
<!DOCTYPE html>
<html>
<head>
<meta charset="utf-8">
<title>Style Placement</title>
<link rel="stylesheet" href="style.css">
<style>

h2 {
  color: maroon;
}

</style>
</head>
<body>
<h1>Style Placement</h1>
<h2>Subheading 1</h2>
<p>Lorem ipsum dolor sit amet, consectetur adipisicing
<h2>Subheading 2</h2>
<p style="text-align: center;">I am centered!</p>

</body>
</html>
```

style.css

```
body {
  background-color: gray;
  font-size: 130%;
}
```

Давайте рассмотрим пример.

В этом примере мы указываем в теге style, что каждый h2 в нашем HTML-документе должен иметь темно-бордовый цвет текста.

Однако есть и другие места, где вы можете указать стиль CSS.

```
<p style="text-align: center;">I am centered!</p>
```

Например, вы можете указать стиль CSS непосредственно для элемента, указав атрибут style и объявления CSS непосредственно в теге элемента.

Объявления стилей будет точно такое же, как и раньше, и они также заканчиваются точкой с запятой, и вы можете указать здесь несколько объявлений, просто завершая их точкой с запятой.

Это называется встроенным стилем, и, как вы могли догадаться, это наименее пригодный для повторного использования способ стилизации элементов.

Обычно этот способ стилизации используется для быстрого тестирования, чтобы посмотреть, как будет выглядеть стиль.

Теперь, что, если у меня есть несколько страниц на моем веб-сайте, и я хочу, чтобы все они выглядели одинаково? Это означает, что указание стилей с помощью тега style мне не поможет.

И мне нужен какой-то другой способ указать стили, который является внешним по отношению к HTML-странице.

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

style.css

```
body {
  background-color: gray;
  font-size: 130%;
}
```

Для этого мы можем использовать тег с именем link, где мы сообщаем браузеру, что есть таблица стилей и указываем ее местоположение, используя атрибут href.

И внешние таблицы стилей – это просто файлы со списками правил CSS. И здесь нет никаких особых тегов.

Как вы можете видеть в этом файле style.css, мы указываем для тела документа, что его цвет фона должен быть серым, а размер шрифта должен составлять 130% от размера шрифта по умолчанию.

Реальные веб-сайты почти всегда используют внешние таблицы стилей, что означает, что вы берете все свои стили и помещаете их во внешний файл, а затем связываете его обратно с несколькими HTML-страницами с помощью тега link.

Этот метод не только повторно использует стили CSS, но и способствует единообразному виду всего веб-сайта.

Теперь давайте рассмотрим разрешение конфликтов при определении стилей.

Само название CSS – это каскадные таблицы стилей. То есть каскадирование – это фундаментальная особенность CSS. Это алгоритм, определяющий, как комбинировать значения свойств из разных источников.

И если есть конфликт в стилизации, чтобы понять какое правило CSS выигрывает, нужно понимать этот алгоритм.

Алгоритм каскадирования объединяет четыре концепции.

Это происхождение, или приоритет происхождения, слияние, а также наследование и специфичность.

Сначала давайте рассмотрим первые две концепции – происхождение и слияние.

Когда два CSS объявления конфликтуют, другими словами, они определяют по-разному одно и то же свойство для одного и того же элемента, срабатывает правило приоритета происхождения.

И это очень простое правило – последнее объявление побеждает.

```
<!DOCTYPE html>
<html>
<head>
<meta charset="utf-8">
<title>Cascade of CSS</title>
<link rel="stylesheet" href="external.css">
<style>

p {
 color: maroon;
}

</style>
</head>
<body>
<h1>Origin Example</h1>
<p>The rule is simple: last declaration wins.</p>
<p style="color: black;">If there is no conflict,
declarations merge into one rule.</p>
</body>
</html>
```

external.css

```
p {
 font-size: 130%;
 background-color: gray;
 color: white;
}
```

Теперь, пытаясь выяснить, что такое последнее объявление, вы должны помнить, что HTML обрабатывается последовательно. То есть сверху вниз.

Поэтому, чем ниже объявление на странице, тем выше его приоритет.

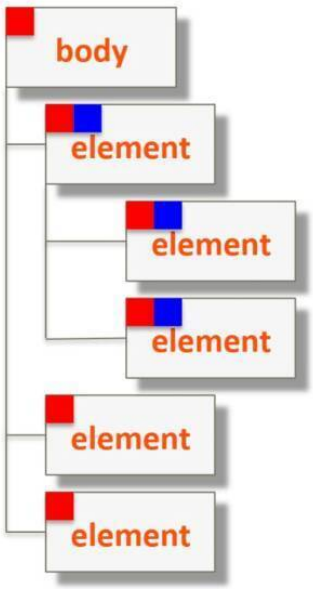
При этом для внешнего CSS думайте, как об объявленном в том месте, где он подвязан.

Теперь о слиянии.

Когда разные объявления CSS не конфликтуют, но свойства CSS для элемента разные, существует простое правило – объявления сливаются.

Таким образом, объявления, например, размера шрифта и объявление цвета, так как это два разных свойства, при том, что они определены для одного и того же элемента, и даже если они определены в разных местах, они сольются в одно объявление. И элемент получит как размер шрифта, так и цвет.

Следующей концепцией, которую мы собираемся рассмотреть, является наследование.



И это довольно простая концепция. Основная идея здесь заключается в том, что у вас есть дерево объектной модели документа. И если вы укажете какое-то свойство CSS для какого-либо элемента, все дочерние элементы этого элемента также наследуют это свойство, и вам не нужно указывать свойство для каждого элемента.

Так, например, если я укажу одно свойство в теге body, каждый элемент, который является дочерним элементом тега body унаследует это свойство. Точно так же, если я укажу свойство для какого-либо элемента на HTML-странице, каждый дочерний элемент этого элемента также унаследует это свойство. И очевидно, что ни один из родителей этого элемента не унаследует это свойство.

Теперь давайте рассмотрим концепцию специфичности.

И у специфичности также есть довольно простое правило – выигрывает наиболее конкретная комбинация селекторов.

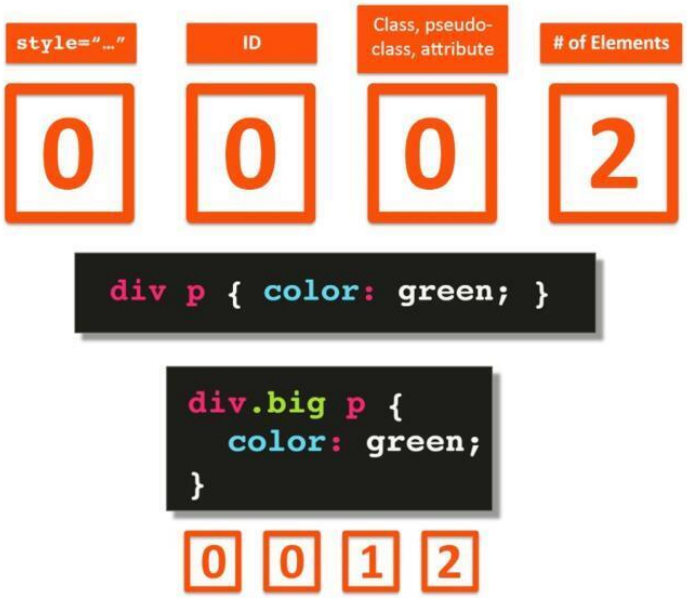
И есть довольно простой метод, который позволяет выяснить, какая комбинация селекторов является более специфичной, чем другие.

Вы можете думать о специфичности как о подсчете очков. Комбинация с наибольшим количеством очков побеждает.

И эту оценку можно рассчитать, если расположить типы факторов, влияющих на оценку, слева направо, причем слева будет наивысшее значение специфичности. Затем просто возьмите правило CSS и заполните эту табличку.

Самое высшее значение специфичности имеет атрибут стиля элемента style. И это происходит, когда вы указываете объявления CSS прямо в элементе, используя атрибут стиля. Нет ничего более конкретного, чем это объявление.

Далее идет идентификатор элемента, затем класс или псевдокласс, а затем количество элементов, которые используются в комбинации.



Здесь у первого правила количество баллов 2, а у второго 12 баллов. Поэтому второе правило выигрывает.

И есть еще одна концепция, которую мы рассмотрим. И это концепция переопределения всех правил с помощью «!important».

```
<!DOCTYPE html>
<html>
<head>
<style>
#myid {
  background-color: blue;
}

.myclass {
  background-color: gray;
}

p {
  background-color: red !important;
}
</style>
</head>
<body>

<p>This is some text in a paragraph.</p>

<p class="myclass">This is some text in a paragraph.</p>

<p id="myid">This is some text in a paragraph.</p>

</body>
</html>
```

В этом примере все три абзаца получат красный цвет фона, хотя селектор идентификатора и селектор класса имеют более высокую специфичность. Потому что правило «!important» переопределяет свойство background-color в обоих случаях.

Правило «!important» говорит что не имеет значение, какая есть специфичность и переопределяет все CSS объявления.

Теперь я хочу предупредить вас об использовании этого «!important». Хотя очень заманчиво пропустить понимание всех этих каскадных правил и правил специфичности и просто хлопать «!important» везде, когда что-то не получается, это очень быстро в реальном проекте

превратится в кошмар обслуживания, когда вы будете переопределять одно «!important» объявление другим «!important» объявлением, и возникнет гигантский беспорядок.

Теперь, давайте поговорим о стилизации текста.

Существует множество свойств CSS, влияющих на отображение текста. Мы не будем пытаться охватить каждое из них. Вместо этого мы рассмотрим несколько свойств, которые иллюстрируют концепции, лежащие в их основе.

```
<!DOCTYPE html>
<html>
<head>
<meta charset="utf-8">
<title>Styling Text</title>
<style>
.style {

}
</style>
</head>
<body>
<p>You can get commonly used font combinations from
<a href="http://www.w3schools.com/cssref/css_websafe_fonts.asp">
http://www.w3schools.com/cssref/css_websafe_fonts.asp</a>. Lorem ipsum dolor sit amet,
consectetur adipisicing elit. Quisquam quod necessitatibus a ullam dolorum amet reprehenderit sit
laudantium reiciendis aperiam. </p>
<p class="style">Lorem ipsum dolor sit amet, consectetur adipisicing elit. Laboriosam fugiat
repudiandae fugit porro commodi.</p>
</body>
</html>
```

В этом примере мы создаем класс с именем style, и мы применяем этот класс ко второму абзацу в нашем html-файле.

И наша задача состоит в том, чтобы стилизовать второй абзац.

И первое, что мы обычно хотим сделать, это указать семейство шрифтов. И здесь указана ссылка на часто используемые шрифты.

- Arial (sans-serif)
- Verdana (sans-serif)
- Helvetica (sans-serif)
- Tahoma (sans-serif)
- Trebuchet MS (sans-serif)
- Times New Roman (serif)
- Georgia (serif)
- Garamond (serif)
- Courier New (monospace)
- Brush Script MT (cursive)

Таким образом, семейство шрифтов – это имя свойства font-family.

```
<style>
.style {
 font-family: Arial, Helvetica, sans-serif;
 color: #0000ff;
 font-style: italic;
 font-weight: bold;
 font-size: 24px; /* points (pt) are used in print, not screen.*/
 text-transform: uppercase;
 text-align: right;
}
</style>
```

И значение для семейства шрифтов может варьироваться. То, что вы обычно указываете в семействе шрифтов, – это комбинация шрифтов. И причина, по которой вы указываете не один шрифт, а несколько шрифтов, заключается в том, что, когда вы указываете семейство шрифтов, вы полагаетесь на компьютер пользователя.

И возможно, что конкретный шрифт не установлен на компьютере пользователя, поэтому вы указываете несколько вариантов.

Свойство font-family должно содержать несколько имен шрифтов в качестве запасных, чтобы обеспечить максимальную совместимость между браузерами и операционными системами. Начните с нужного шрифта и закончите общим семейством, чтобы позволить браузеру выбрать аналогичный шрифт в общем семействе, если другие шрифты недоступны. Названия шрифтов должны быть разделены запятой.

И на тот случай, если вы не знаете разницы между Serif и Sans-Serif, скажу, что есть шрифты без засечек и шрифты с засечками – это шрифты, в которых есть не только линии, но и небольшое украшение в конце.

Далее давайте изменим цвет. И это мы делаем с помощью свойства color.

И здесь можно использовать предопределенные имена цветов red, green, blue и т.д., а можно использовать шестнадцатеричное значение для определенного цвета.

Свойство font-style указывает, хотите ли вы, чтобы текст был курсивным или обычным.

```
<style>
p.normal {
  font-style: normal;
}

p.italic {
  font-style: italic;
}

p.oblique {
  font-style: oblique;
}
```

This is a paragraph in normal style.

This is a paragraph in italic style.

This is a paragraph in oblique style.

Следующее свойство – это вес шрифта.

```
<style>
p.normal {
  font-weight: normal;
}

p.light {
  font-weight: lighter;
}

p.thick {
  font-weight: bold;
}

p.thicker {
  font-weight: 900;
}
</style>
```

This is a paragraph.

This is a paragraph.

This is a paragraph.

This is a paragraph.

И вес шрифта можно указать от нормального до жирного. И вы также можете указать вес с помощью числа.

Далее мы можем указать размер шрифта.

```
h1 {
  font-size: 40px;
}

h1 {
  font-size: 2.5em; /* 40px/16=2.5em */
}


body {
  font-size: 100%;
}

h1 {
  font-size: 2.5em;
}
```

```
<h1 style="font-size:10vw">Hello World</h1>
```

Почти каждый браузер имеет размер шрифта по умолчанию 16 пикселей. Не путайте это с точками pt, здесь используются пиксели. Точки pt используются в печати, но не на экране. На экране вы используете пиксели. Пиксели считаются абсолютной единицей измерения размера. Тем не менее, у них есть относительная составляющая.

Пиксели привязаны к устройству просмотра. Для устройств с низким DPI или низким количеством точек на дюйм один пиксель соответствует одной точке пикселя устройства на дисплее. Для принтеров и экранов с высоким разрешением один пиксель подразумевает несколько точек пикселей устройства.

Дело в том, что устройства с более высоким DPI дадут вам более четкий текст, потому что для каждого пикселя, который он рисует, он фактически рисует несколько точек пикселей на устройстве.

При всем при этом пиксели по-прежнему считаются абсолютной единицей измерения.

Свойство text-transform управляет преобразованием текста элемента в заглавные или прописные символы.

Другое полезное свойство – text-align, которое позволяет центрировать или выравнивать текст по правому и левому краю.

Теперь давайте немного поговорим об относительном размере шрифта. И здесь есть две относительные единицы измерения, это проценты и ems.

```
body {
  font-size: 100%;
}
```

Здесь тег body имеет размер 120%. И это означает, что мы хотим взять размер, который браузер предоставляет по умолчанию, и увеличить его на 120%.

В большинстве браузеров размер текста по умолчанию составляет 16 пикселей. Таким образом, 120% от 16 пикселей будет чуть больше 19 пикселей.

```
h1 {
  font-size: 2.5em;
}
```

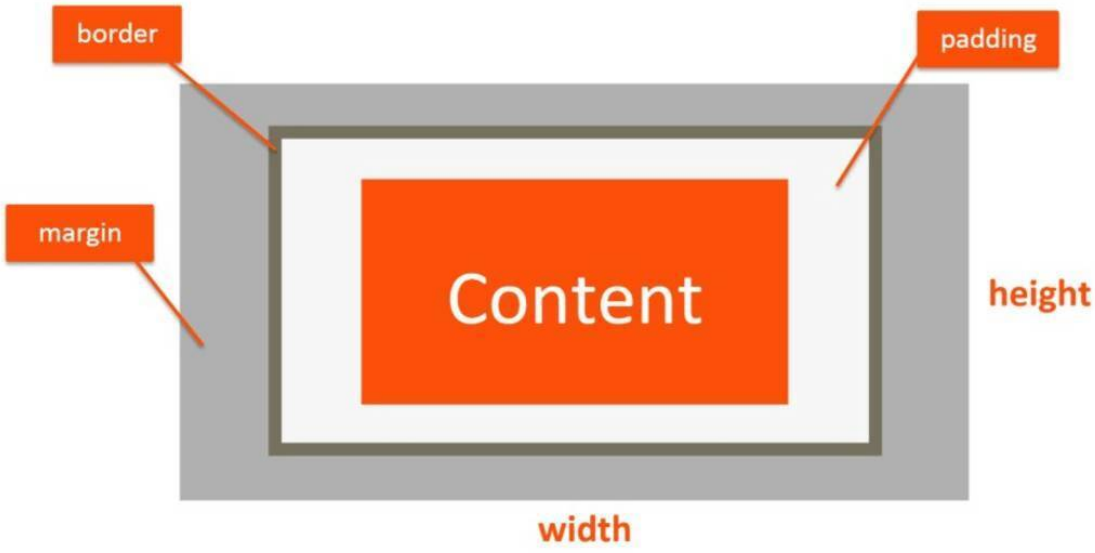
Здесь размер текста составляет 2em. И это единица измерения, эквивалентная ширине буквы в этом конкретном шрифте, который мы используем.

Поначалу это звучит немного запутанно, но суть в том, что это относительный размер.

Так как мы установили размер шрифта 120% во всем теле документа, и когда вы комбинируете этот размер шрифта с 2.5em, это говорит о том, что вы хотите увеличить шрифт в два с половиной раза по сравнению со 120%.

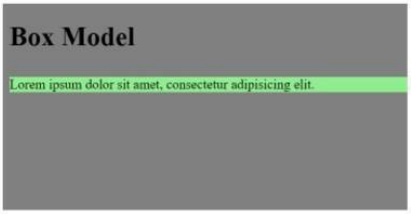
И наконец размер текста можно задать с помощью единицы vw, что означает «ширину области просмотра» viewport . Viewport – это размер окна браузера и 1vw = 1% ширины области просмотра. Если окно просмотра имеет ширину 50 см, 1vw составляет 0,5 см.

Теперь, в HTML каждый элемент считается блоком. Однако этот блок содержит не просто контент.



Помимо содержимого, блок содержит отступы, границы и поля. И существуют правила, определяющие, как эти компоненты блока влияют на компоновку, а также, как вычисляются ширина и высота блока.

```
<style>
body {
  background-color: gray;
}
#box {
  background-color: blue;
}
#content {
  background-color: #90EE90; /*green*/
}
h1 {
  margin-bottom: 30px;
}
</style>
</head>
<body>
<h1>Box Model</h1>
<div id="box">
  <div id="content">Lorem ipsum dolor sit amet, consectetur adipisicing elit. </div>
</div>
```



В качестве примера давайте рассмотрим структуру HTML этого документа.

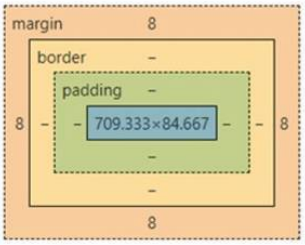
Здесь есть тег h1 и тег div с идентификатором box, а внутри этого div есть еще один div с идентификатором content и текстом.

И так как div является элементом блочного уровня, он пытается заполнить родительский элемент целиком по ширине.

Но если приглядеться, здесь есть некоторое пространство между заголовком и текстом.

Давайте откроем инструменты разработчика Chrome и выясним, что это за пространство.

```
body {                                          таблица стилей агента пользователя
   display: block;
   margin: ▸ 8px;
}
```



Давайте выберем тег body и посмотрим, что происходит.

И мы видим, что тег body имеет отступ около восьми пикселей. Откуда же взялись эти восемь пикселей?

И здесь написано таблица стилей пользовательского агента. Это означает, что это сам браузер. Это стили браузера по умолчанию.

Мы можем сделать margin 0, и когда мы это сделаем, мы увидим, что теперь наш контент находится на одном уровне с фактическими границами окна браузера.

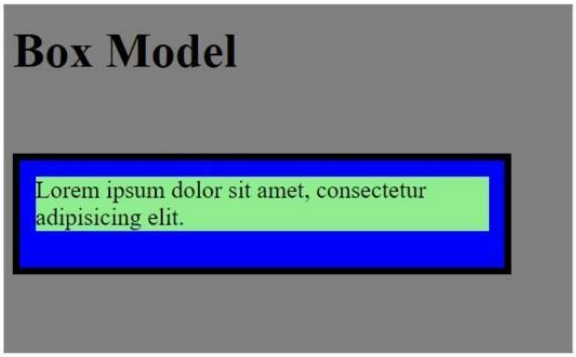
Далее вы можете видеть, что контекст зеленый. Но мы не видим фон блока, в котором на самом деле находится контент, потому что мы бы видели что-то синее на заднем плане.

И это потому, что внутренний блок содержимого полностью закрывает внешний, потому что они имеют одинаковый размер. Они оба пытаются заполнить своего родителя, который в данном случае является тегом body.

Поэтому давайте установим некоторые отступы внешнего блока.

```
#box {
  background-color: blue;
  padding: 10px;
  border: 5px solid black;
  width: 300px;
  height: 50px;
  margin-top: 50px;
  overflow: auto;
}
```

Давайте установим padding 10 пикселей сверху, 10 пикселей справа, 10 пикселей внизу и 10 пикселей слева.



И вуаля, теперь мы увидели синий фон. Также мы добавили этому блоку черную границу с толщиной 5 пикселей и размеры самого блока 300 пикселей на 50 пикселей, а также отступ сверху от заголовка 50 пикселей.

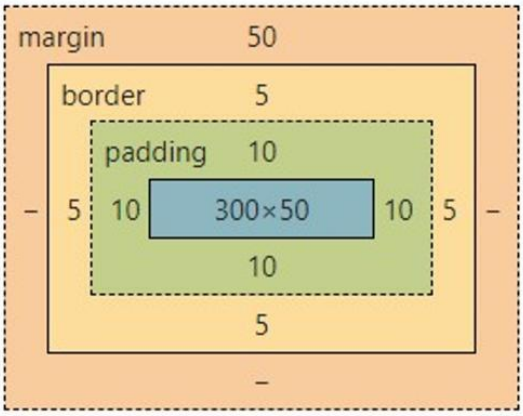
Кроме того, мы добавили свойство overflow. Свойство overflow указывает, следует ли обрезать содержимое или добавить полосы прокрутки, когда содержимое элемента слишком велико и не помещается в указанной области. И свойство overflow имеет следующие значения:

visible – по умолчанию. Переполнение контента в блоке не отсекается, и содержимое отображается за пределами блока.

hidden – переполнение контента обрезается, и остальное содержимое будет невидимым.

scroll – переполнение контента обрезается, и добавляется полоса прокрутки, чтобы увидеть остальную часть содержимого.

auto – аналогично прокрутке, но добавляет полосы прокрутки только при необходимости.



В результате мы можем увидеть, что наш блок имеет размеры не 300 пикселей на 50 пикселей, а здесь добавляется отступ, ширина границы и поле. Так что, когда вы устанавливаете размеры блока, нужно учитывать и все остальные свойства. Здесь на самом деле мы устанавливаем не ширину блока, а ширину содержимого.

```
* {
  box-sizing: border-box;
  margin: 0;
  padding: 0;
}
```

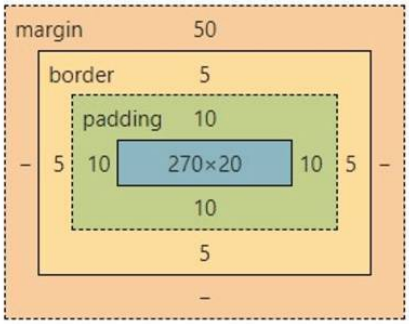
В CSS 3 есть такое свойство как box-sizing со значением border-box.

Свойство CSS box-sizing позволяет нам включать отступы и границы в общую ширину и высоту элемента.

Без свойства CSS box-sizing, по умолчанию ширина и высота элемента вычисляются следующим образом:

ширина + отступ + граница = фактическая ширина элемента

высота + отступ + граница = фактическая высота элемента



С этим значением border-box свойства box-sizing все будет включено и размер нашего блока станет 300 пикселей на 50 пикселей.

Конец ознакомительного фрагмента.