**Атрибуты**

Атрибуты — это свойства тега. С помощью них мы задаём параметры тега.  
  
Сразу возьмём пример: тег <a> — ссылка. Для задания адреса, куда будет вести эта ссылка, нам понадобится атрибут href. Вот так будет выглядеть ссылка на страницу ITC Вконтакте:

<a href="https://vk.com/itc.digital">ITC Вконтакте</a>

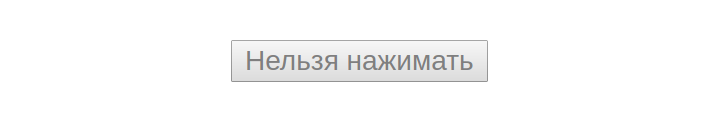
Атрибут указывается внутри тега, значение атрибута указывается внутри кавычек. Атрибуты отделяются друг от друга пробелами. Пример ссылки на страницу ITC, которая откроется в новой вкладке:

<a href="https://vk.com/itc.digital" target="\_blank">ITC Вконтакте</a>

У атрибута может не быть значения, тогда наличие атрибута включает какой-то параметр, а отсутствие - отключает. Например, атрибут disabled. Если кнопке <button> задать атрибут disabled, она станет серой и на неё невозможно будет нажать.

<button disabled>Нельзя нажимать</button>

Результат:



В VSCode можно быстро вставить шаблон HTML-документа. Для этого нужно открыть пустой .html файл, написать html:5 и нажать Tab.

**Элементы и их виды**

Элементы - то, что создаётся тегами. Можно сказать, что теги это текстовое представление элементов. Элементы бывают двух видов:

**Блочные элементы**

Составляют структуру страницы.

**Особенности**:

* блоки располагаются друг под другом по вертикали
* запрещено вставлять блочный элемент внутрь строчного
* занимают всё допустимое пространство по ширине
* высота вычисляется автоматически, исходя из содержимого

**Примеры**:

* абзацы <р>
* списки: маркированные (с маркером) <ul> и нумерованные (с числами) <ol>
* заголовки: от первого уровня <h1> до шестого уровня <h6>
* статьи <article>
* разделы <section>
* длинные цитаты <blockquote>
* блоки общего назначения  <div>

**Строчные элементы**

Используются для форматирования текстовых фрагментов. Обычно содержат одно или несколько слов.

**Особенности**:

* элементы, идущие подряд, располагаются на одной строке и переносятся на другую при необходимости
* внутрь допустимо вставлять текст или другие строчные элементы, помещать блочные элементы - запрещено

**Примеры**:

* ссылки <a>
* выделенные слова <em>
* важные слова <strong>
* короткие цитаты <q>
* аббревиатуры <abbr>

**Если ты запомнишь только одну вещь из этого урока, запомни, что:**

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

**Списки**

В HTML существует три вида списков:

**Маркированный**

Список из неупорядоченных элементов.

Состоит из двух тегов:

* <ul> (*unordered list*) - тег начала и конца списка
* <li> (*list item*) - пункт списка

Пример:

Список ингредиентов:

<ul>

 <li>Картошка</li>

 <li>Морковка</li>

 <li>Свекла</li>

</ul>

**Нумерованный**

Упорядоченный список, каждый пункт имеет свой номер.

Состоит из двух тегов:

* <ol> (*ordered list*) - тег начала и конца списка
* <li> (*list item*) - пункт списка

Пример:

Инструкция по приготовлению:

<ol>

 <li>Довести воду до кипения</li>

 <li>Засыпать ингредиенты</li>

 <li>Варить 10 минут</li>

</ol>

**Список определений**

Используются для создания списка терминов и их определений. В общем случае, каждый пункт — это пара "имя/значение".

Состоит из трёх тегов:

* <dl> (d*escription list*) - тег начала и конца списка
* <dt> (*term*) - термин
* <dd> (*description*) - определение

Пример:

<dl>

 <dt>Гаспачо</dt><dd>лёгкий холодный суп из перетёртых в пюре свежих овощей</dd>

 <dt>Том-ям</dt><dd>кисло-острый суп на основе куриного бульона с креветками, курицей, рыбой или другими морепродуктами</dd>

 <dt>Борщ</dt><dd>разновидность супа на основе свёклы, которая придаёт борщу характерный красный цвет</dd>

</dl>

**Изображения**

Для добавления изображения используется тег <img>. Это одинарный тег. Вот его основные атрибуты:

* src - ссылка на картинку﻿
* alt - текст, который отображается вместо картинки, если она не загрузилась
* title - текст, который отображается при наведении мыши на картинку
* width - ширина картинки в пикселях
* height - высота картинки в пикселях

**Пример:**

<img

 src="http://example.com/cat.jpg"

 title="Мурка"

 alt="Рыжая кошка валяется в снегу"

 width="640"

 height="480"

>

**Семантичные изображения с подписью в HTML 5**

В HTML 5 появились теги для оформления объектов ﻿с подписями﻿ - figure и figcaption. Если твоей картинке нужна подпись - пользуйся ими. Пример кода:

<figure>

<img src="https://www.google.ru/images/branding/googlelogo/2x/googlelogo\_color\_120x44dp.png﻿">

<figcaption>

Лого гугла от ﻿2015 года

</figcaption>

﻿</figure>

**Ссылки и адреса**

Ты уже знаком со ссылками:

<a href="https://google.com/">Google</a>

Повторим: для создания ссылки необходимо использовать тег <a>. Атрибут **href** указывает адрес, по которому будет совершён переход.

Адреса бывают двух видов:

**Абсолютные адреса**

Абсолютный адрес, записанный в полной форме. Например,

https://google.com/doodles

Давай разберём этот адрес:

* https - так называемая «схема», обычно это название протокола. HTTPS - защищённая версия HTTP
* google.com - доменное имя сайта
* /doodles - путь (директория) внутри сайта

Ещё пример:

file:///C:/Users/admin/Desktop/Новая%20папка/image.jpg

* file - схема URI, предназначенная для того, чтобы адресовать файлы на локальном компьютере или в локальной сети ([подробнее на Википедии](https://ru.wikipedia.org/wiki/File_(%D1%81%D1%85%D0%B5%D0%BC%D0%B0_URI)))
* */C:/Users/admin/Desktop/Новая%20папка/image.jpg* - путь до файла. *%20* - код пробела в [URI-кодировании](https://ru.wikipedia.org/wiki/URL#%D0%9A%D0%BE%D0%B4%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5_URL).

**Относительные адреса**

Относительный - сокращённый адрес. В таком адресе начальная часть опущена и браузер использует текущий адрес для определения полного адреса. Примеры:

* //google.com - ссылка на домен в текущем протоколе: если мы находимся по адресу, который начинается с http, то ссылка будет вести на [http://google.com](http://google.com/)
* /sheets - ссылка на путь внутри текущего домена: если мы находимся на [http://google.com](http://google.com/), то ссылка будет вести на <http://google.com/sheets>, а если на [http://facebook.com](http://facebook.com/), то на <http://facebook.com/sheets>.
* page2 - ссылка на путь внутри текущей директории: если мы находимся на [http://site.com/routes/page1](http://site.com/routes/.../.../page1), то попадём на [http://site.com/routes/page2](http://site.com/routes/.../.../page2)

**Пример использования относительного адреса**

Файловая система:

Новая папка

├───img

│ ├───kisa.jpg

│ └───kot.png

├───index.html

└───style.css

Код в index.html:

...

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

...

<img src="img/kisa.jpg">

<img src="img/kot.png">

**При выполнении заданий с использованием файлов - картинок, шрифтов, веб-страниц, которые находятся локально (то есть у тебя на устройстве), используй относительные ссылки. Потому что при загрузке кода на сервер, ссылке вроде file:///C:/Users/admin/Desktop/Новая%20папка/image.jpg перестанут работать.**

**Якоря**

Ссылки могут ссылаться не только на страницу, но ещё и на конкретное место на странице. Такие ссылки называются *якорными,*а места, на которые они ссылаются*-- якорями.*

Якорному элементу нужно прописать атрибут **id**c именем якоря, например:

<h1 id="anchor">Якорь</h1>

Теперь, чтобы сослаться на этот якорь, нужно использовать вот такую ссылку:

<a href="#anchor">Перейти к якорю</a>

При нажатии на ссылку произойдёт переход к нужному месту страницы, а в адресной строке к адресу страницы добавится адрес якоря **#anchor**

Якоря могут использоваться как в относительных ссылках, как в примере выше, так и в абсолютных, например: <http://example.com/page#anchor>

**Таблицы**

**Таблицы** в HTML создаются при помощи тега <table>.

Внутри него размещают **строки таблицы** <tr> (*table row*)

Внутри строк помещают **ячейки строки** <td> (*table data*).

Тегом <th> (*table header*) размечаются **заголовочные ячейки**. Он отличается от <td> тем, что его содержимое будет выделено полужирным и выровнено по центру.

**Пример таблицы**:

<table>

  <tr>

    <th>Имя</th>

    <th>Возраст</th>

    <th>Суперспособность</th>

  </tr>

  <tr>

    <td>Логан</td>

    <td>186</td>

    <td>Повышенная способность к регенерации</td>

  </tr>

  <tr>

    <td>Профессор икс</td>

      <td>94</td>

    <td>Чтение мыслей, вызов иллюзий, временного паралича, способность останавливать время</td>

  </tr>

</table>

**Результат:**



По умолчанию границы таблиц не отображаются. Чтобы включить отображение границ, нужно использовать атрибут **border**.

**Важно!!!**

**Атрибут border устарел** и его применение в HTML5 не приветствуется. Используйте данный атрибут только пока знакомитесь с таблицами. Данный атрибут заменяет свойство border в CSS.

Пример:

<table border="1"> ... </table>

**thead, tbody, tfoot, caption**

Существует возможность группировать строки таблицы тегами <thead>, <tfoot>, <tbody>. Они не являются обязательными, но их рекомендуется использовать. Они дают частям таблицы семантический смысл.

<thead> - для заголовка,  
<tfoot> - нижний колонтитул таблицы,  
<tbody> - основной контент таблицы

Интересный факт: <tfoot> в коде можно расположить перед <tbody> или после него, но браузеры всегда выводят его в конце таблицы.

<thead> и <tfoot> можно использовать только по одному разу в одной таблице, а количество <tbody> может быть любым. С помощью нескольких <tbody> можно разделить контент таблицы на смысловые части. Например, так в таблице могут быть представлены данные за несколько лет, и каждый год будет вынесен в отдельный <tbody>.

Название таблицы можно разместить в теге <caption>. Этот тег располагают внутри тега <table> в самом начале, перед тегами <thead>, <tfoot>, <tbody>

**Пример использования тегов**

<table border="1">

  <caption>Сотрудники отдела поддержки</caption>

  <thead>

    <tr>

      <th>Имя</th>

      <th>Должность</th>

    </tr>

  </thead>

  <tbody>

    <tr>

      <td>Петя</td>

      <td>Менеджер</td>

    </tr>

    <tr>

      <td>Маша</td>

      <td>Разработчик</td>

    </tr>

  </tbody>

  <tfoot>

    <tr>

      <td>Вася</td>

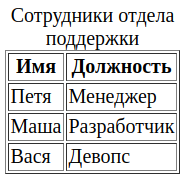
      <td>Девопс</td>

    </tr>

  </tfoot>

</table>

**Результат:**



**Объединение столбцов**

Если количество столбцов в одной строке меньше, чем количество столбцов в другой строке таблицы, то в строке с меньшим количеством ячеек образуется пустое пространство.

**Пример:**

<table border="1">

  <tr>

    <td>Первая ячейка</td>

    <td>Вторая ячейка</td>

    <td>Третья ячейка</td>

  </tr>

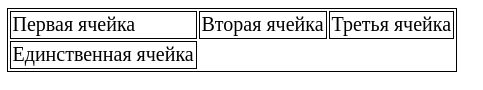
  <tr>

    <td>Единственная ячейка</td>

  </tr>

</table>

**Результат:**



Как видно, ячейка не занимает свободное пространство. Чтобы это исправить, нужно растянуть её вправо с помощью атрибута **colspan**, и значением указать количество столбцов, которое в сумме будет занимать ячейка.

**Пример:**

<table border="1">

  <tr>

    <td>Первая ячейка</td>

    <td>Вторая ячейка</td>

    <td>Третья ячейка</td>

  </tr>

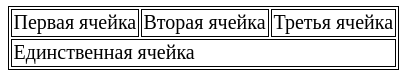
  <tr>

    <td colspan="3">Единственная ячейка</td>

  </tr>

</table>

**Результат:**



**Объединение строк**

Для объединения строк по аналогии со столбцами используется атрибут **rowspan**. Только в данном случае, ячейка растягивается вниз на указанное количество строк.

**Пример:**

<table border="1">

  <tr>

    <td rowspan="2">Левая ячейка</td>

    <td>Верхняя ячейка</td>

  </tr>

  <tr>

    <td>Нижняя ячейка</td>

  </tr>

</table>

**Результат:**



**Объединение строк и столбцов можно сочетать:**

<table border="1">

  <tr>

    <td rowspan="2">Левая ячейка</td>

    <td>Средняя ячейка</td>

    <td>Правая ячейка</td>

  </tr>

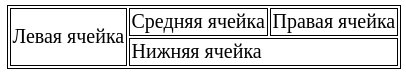
  <tr>

    <td colspan="2">Нижняя ячейка</td>

  </tr>

</table>

**Результат:**



**Формы**

Формы используются для сбора информации, которую пользователь вводит в специально отведённые поля этой формы. Когда он введёт свои данные и нажмет кнопку «Отправить», все эти данные будут отправлены на сервер. Затем они будут обработаны и сервер отправит ответ пользователю. Существует два основных метода отправки данных: **GET и POST**.

**GET**

Когда ты вводишь в адресной строке браузера какой-либо адрес и переходишь по нему, ты отправляешь серверу запрос, называемый GET. В таком запросе данные могут отсутствовать, как здесь: <https://www.google.ru/>. А вот запрос [https://www.google.ru/search?q=itс+stepik](https://www.google.ru/search?q=it%D1%81+stepik) содержит в себе переменную q, которая имеет значение itc stepik. В данном случае запрос отправляется на адрес <https://www.google.ru/search>,  а данные из полей и их названия идут после **?** через знак **&**. Предварительно данные кодируются в URL код, чтобы сервер не перепутал служебные символы (вроде **/**, или **﻿&**) с частью запроса.

**POST**

Методом POST так же можно отправлять данные в URL. Но, в отличие от GET, он может иметь *тело -* специальную "коробочку", в которую можно положить данные, которые уйдут на сервер.

Этот метод обычно используется для отправки форм и загрузки файлов.

*Кроме GET и POST существуют методы PUT, PATCH и DELETE. Они отличаются от POST только названием и разработчики выбирают тот или иной метод исходя из смысла действия.*

Страница, созданная методом GET, может быть открыта повторно множество раз. Такая страница может быть кэширована браузерами, проиндексирована поисковыми системами и добавлена в закладки пользователем. Из этого следует, что метод GET следует использовать для получения данных от сервера и не желательно в запросах, предполагающих внесений изменений в ресурс. Например, можно использовать метод GET в HTML форме фильтра товаров: когда нужно, исходя из данных введенных пользователем, переправить его на страницу с отфильтрованными товарами, соответствующими его выбору. Запрос, выполненный методом POST, напротив следует использовать в случаях, когда нужно вносить изменение в ресурс (выполнить авторизацию, отправить форму оформления заказа, форму обратной связи, форму онлайн заявки). Повторный переход по конечной ссылке не вызовет повторную обработку запроса, так как не будет содержать переданных ранее параметров. Метод POST имеет большую степень защиты данных, чем GET: параметры запроса не видны пользователю без использования специального ПО, что дает методу преимущество при пересылке конфиденциальных

**Создание формы**

Элемент формы создаётся парным тегом <form>. Внутри размещаются сами поля, причем те, что расположены за пределами элемента, отправлены не будут. При необходимости в элементе формы можно использовать стороннюю разметку. Тип отображения по умолчанию block (блочный элемент). Данный тег имеет два обязательных атрибута: action и method.

В атрибуте action указывается ссылка на обработчик формы. Обработчиком формы является PHP файл (или файл, написанный на других языках программирования), который будет обрабатывать данные этой формы.

Атрибут method предназначен для указания метода отправки данных на сервер (GET или POST).

**<input>**

Поле задается одинарным тегом <input>. Данный тег стилизуется браузерами по-разному. Тип отображения по умолчанию inline (встроенный элемент), поэтому все поля отображаются в одной строке. Эта проблема решается подключением стилей, а пока мы до них не дошли - используй тег <br>.

Если кому пример нужен, вот:

<title>Один из возможных вариантов HTML формы</title>

</head>

<body>

<form name="f1" method="get" action="enter\_data.php">

<input name="link" type="hidden" value="index.php" />

Логин: <br />

<input name="login" type="text" size="25" maxlength="30" value="Вася" /> <br />

Пароль: <br />

<input name="pd" type="password" size="25" maxlength="30" value="" /> <br />

<input name="remember" type="checkbox" value="yes" /> Запомнить <br />

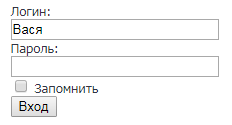
<input type="submit" name="enter" value="Вход" />

</form>

</body>

</html>

Выглядеть будет так:



Вот здесь есть подробное описание:

<http://ab-w.net/HTML/html_forms.php>

**Типы полей форм**

От пользователя нам могут понадобиться самые разные данные. Для удобства их сбора появились типы полей форм. Они задаются через атрибут ***type***. Вот основные:

* *Текстовое поле.* Все просто: в атрибуте пишем ***text***.

<input type="text">



* *Ввод пароля*. Указывается значение ***password***.
* <input type="password">



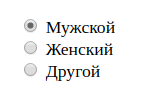
* *Флаги (чекбоксы).* Они используются для выбора нескольких вариантов ответа и обозначаются в атрибуте как ***checkbox***. Подпись к галочке нужно писать отдельно.

<input type="checkbox">



* *Радио-кнопки.*Это выбор одного из нескольких вариантов. Для создания в атрибуте прописываем radio. Переключатели между собой нужно связывать: пишем атрибут name с одинаковыми значениями. Также необходим атрибут value.
* <input type="radio" name="gender" value="male" checked> Мужской<br>
* <input type="radio" name="gender" value="female"> Женский<br>

<input type="radio" name="gender" value="other"> Другой



Эти типы создают кнопки:

* *Загрузка файлов.*При значении атрибута file появляется кнопка загрузки.

<input type="file">



* *Сброс всех значений*. Для создания такой кнопки пишем reset. Чтобы все сработало, кнопка должна находиться внутри формы, в которой нужно сбросить значения.

<input type="reset">



* *Отправка.*Любая форма бессмысленна, если не отправляется на сервер. Значение для такой кнопки: submit.

<input type="submit">

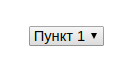


* *Просто кнопка.*Чтобы создать кнопку без определенного действия указываем button.
* <button>Нажми на меня</button>



Фух! Очень много этих полей, да? Осталось последнее - раскрывающийся список. Это тег <select>. У него есть несколько атрибутов: ***name, size, multiple***. Атрибут ***name*** работает как у переключателей. **Size** отвечает за то, сколько строк списка будет одновременно отображено. В качестве значения туда пишется целое положительное число. Атрибут ***multiple*** отвечает за то, можно ли выбрать сразу несколько вариантов из списка (как checkbox); это атрибут без значения.

Каждый элемент списка выделяется в тег <option>. Так же, как и с <input type="checkbox"> и <input type="radio"> нужно указывать атрибут value со значением, которое будет отправлено на сервер.



**Новые типы полей**

С появлением стандарта HTML5 добавились и новые типы полей форм. В первую очередь они стали гораздо удобнее для пользователей.

* **Ввод E-Mail***, type="email".*Текстовое поле, у которого на клавиатуре мобильных устройств появляется символ @.
* **Номер телефона**, *type="tel".*На мобильных устройствах открывается клавиатура с числами.
* **Ввод ссылки**, *type="url".*
* **Числовое поле**, *type="number".*Помимо клавиатуры с цифрами появляется возможность переключать значения поля. Атрибуты **min** и **max** определяют нижнее и верхнее возможное значение, **step** - шаг изменения, a **value**- начальное значение.
* **Числовой ползунок**, *type="range"*. Появляются уже указанные атрибуты **min**, **max**, **step**и **﻿value.**
* **Поиск**, *type="search"*. Google Chrome добавляет крестик для очистки введенной строки. На мобильных устройствах появляется кнопка поиска.

**Через внешний файл (link)**

В *head* HTML-документа нужно добавить *link* со ссылкой на файл *.css*:

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

Пример [в предыдущем уроке](https://stepik.org/lesson/179559/step/1?unit=154202).

**Через тэг “<style>”**

В *head* HTML-документа нужно добавить тэг “style”:

<style>

 .example {

  background: #ebac0c;

 }

</style>

**Через атрибут “style”**

Чтобы стилизовать конкретный элемент, можно использовать атрибут “style”:

<div style="border: 1px solid #000; margin-top: 16px;">Див со стилем</div>

Синтаксис внутри атрибута совпадает с синтаксисом внутри фигурных скобок в селекторе CSS.

Синтаксис CSS состоит из двух частей: *селекторы* и *свойства*. Селекторами мы указываем элементы, а свойствами описываем их стиль.

**Селекторы**

Селектор это правило, по которому будут выбраны элементы — например, мы можем обратиться ко всем параграфам или картинкам. Как?

p {} img {}

**Свойства**

Свойства пишутся у конкретных селекторов между фигурными скобками в формате ключ: значение.

Сложно? Вот пример как сделать у всех параграфов красный цвет и подчёркивание:

p {

 color: red;

 text-decoration: underline;

}

**Каждое новое свойство пишется с новой строки.**

**После каждого свойства необходимо ставить точку с зяпятой.**

**Типы селекторов**

Это основные типы селекторов. Есть и более сложные, но их очень редко используют даже опытные разработчики. Потому что один из главных принципов в написании кода - [KISS](https://ru.wikipedia.org/wiki/KISS_(%D0%BF%D1%80%D0%B8%D0%BD%D1%86%D0%B8%D0%BF)).

*Каждый селектор имеет свою специфичность – если стили в разных селекторах будут конфликтовать, применятся они будут в порядке специфичности – от более специфичному к менее специфичному.*

*Подробнее об этом на MDN:*[*https://developer.mozilla.org/ru/docs/Web/CSS/Specificity*](https://developer.mozilla.org/ru/docs/Web/CSS/Specificity)

**По тегу**

Самый простой селектор - выбирает элементы по их тегу:

h1 {

/\* стили для всех h1 \*/

}

Этот селектор выберет все элементы *<h1>.*

**По классу**

Самый часто используемый селектор — по классу. Задаём в HTML класс элементам, к которым применить стиль:

<div class="card">

 Карточка

</div>

И теперь эти элементы можно выбрать по имени класса. **Имя селектора начинается с точки:**

.card {

/\* стили для всех элементов с class="card" \*/

background: #333; /\* фон серого цвета \*/

}

**По id**

Задавать стили по id - дурной тон, старайся его избегать. Тут всё тоже самое, что и с классом, только атрибут называется id:

<button id="button-go-to-top">

 Наверх

</button>

**И имя селектора начинается с решётки**:

#button-go-to-top {

/\* стили для элемента с id="button-go-to-top" \*/

text-decoration: underline; /\* подчёркивание текста \*/

}

**По атрибуту**

Не самый популярный селектор, но иногда он полезен:

<button data-my-custom-attribute="my-custom-value">

 Нажми на меня

</button>

 Имя и значение атрибута пишется в квадратных скобках. Работает с любым атрибутом:

[data-my-custom-attribute="my-custom-value"] {

color: red;

}

*Подробнее об этом селекторе на MDN:*[*https://developer.mozilla.org/ru/docs/Web/CSS/Attribute\_selectors*](https://developer.mozilla.org/ru/docs/Web/CSS/Attribute_selectors)

**Любой элемент**

Селектор \* выбирает абсолютно любой элемент. Самый непопулярный селектор, обычно используется для костылей. Вряд ли он будет часто тебе нужен. Просто знай, что он есть.

\* {

margin: 0;

}

/\* абсолютно всем элементам будет установлен margin: 0 \*/

**Составные селекторы**

Составные селекторы состоят из комбинации простых.

**Группировка селекторов**

Один и тот же набор свойств можно применить к разным селекторам. Пример:

button,

.button,

.cta-button {}

Нужно просто указать  через запятую все селекторы, к которым ты хочешь применить стили.

**Элемент с классом**

Можно стилизовать конкретный элемент, если у него есть определённый класс. Примеры:

p.example {}

Селектор выберет все p, у которых есть класс example.

.main.active {}

Селектор выберет все элементы с классом main, у которых также есть и класс active. Пример такого элемента:

<div class="main active">Пример</div>

**Вложенность**

Эти селекторы используются для выбора элементов, которые вложены в другие элементы.

**Вложенность на любом уровне**

**Если между двумя простыми селекторами стоит пробел и ничего больше, это означает вложенность на любом уровне**. Пример:

.page p {

 text-decoration: underline;

}

Этот селектор применит подчёркиваение ко всем <p>, которые находятся внутри элемента с классом *page*на любом уровне вложенности.

<div class="page">

 <p>Как писал Пелевин:</p>

 <div class="quote">

  <p>

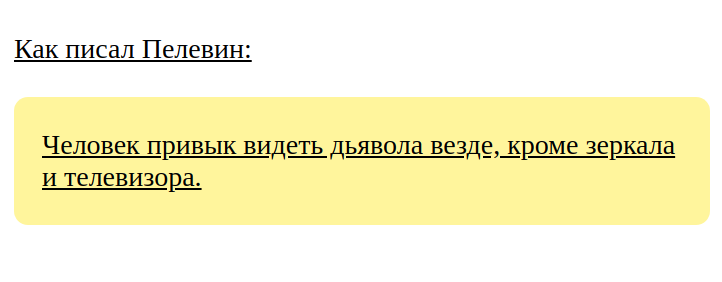
   Человек привык видеть дьявола везде, кроме зеркала и телевизора.

  </p>

 </div>

</div>

Результат:



**Вложенность на первом уровне**

Правая угловая скобка **>**обозначает вложенность на первом уровне. Пример:

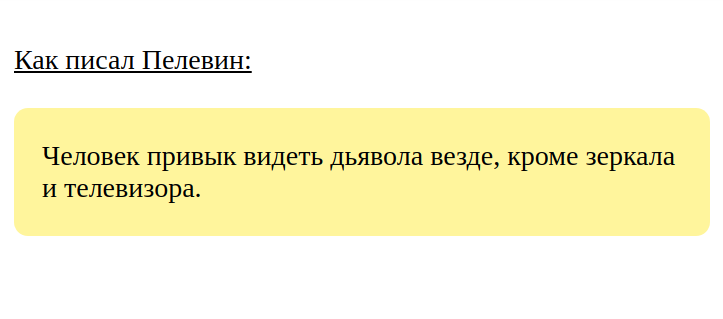
.page > p {

 text-decoration: underline;

}

Этот селектор применится ко всем <p>, которые находятся внутри элемента с классом *page*на первом уровне вложенности.

Для HTML из предыдущего примера:



**Как понимать составные селекторы**

Хороший способ понимать смысл составных селекторов - читать их справа налево.

Давай попробуем разобраться на примере:

.page > .part {}

Этот селектор выберет все элементы **.part**, которые находятся на первом уровне вложенности в **.page**.

*Здесь и дальше, вместо того, чтобы писать «элемент с class="part"», мы будем писать просто****.part****﻿ - точка вначале подразумевает, что это имя класса.*

Ещё пример:

.main .side-menu .menu-item {}

Селектор применится ко всем элементам **.menu-item**, которые находятся внутри элементов **.side-menu**, которые находятся внутри элементов **.main**.

Теперь попробуй самостоятельно:

.promo-bar > .promo-item > .promo-cta-button {}

Селекторы, которые идут дальше, используются редко. Если ты не запомнишь как они работают, ничего страшного - самое главное ты знаешь, что они есть. Если что,  сможешь загуглить их при необходимости или использовать этот урок как справочник.

**Элементы с определенным атрибутом**

Можно задавать стили в зависимости от конкретного значения конретного ﻿атрибута﻿﻿. Пример:

﻿input[type="password"] {}

﻿Этот селектор выберет все элементы <input type="password">

**Первый соседний элемент**

Для того, чтобы *выбрать* элемент, который находится сразу после другого элемента, используется знак **+**. Пример:

img + p {

 margin-top: 0;

 font-style: italic;

}

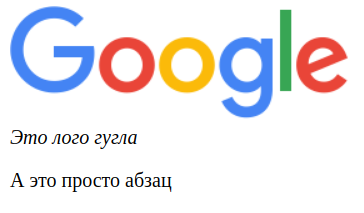
Этот код применит стили во всех <p>, идущих **сразу после** <img>:

<img src="https://www.google.ru/images/branding/googlelogo/1x/googlelogo\_color\_272x92dp.png" />

<p>Это лого гугла</p>

<p>А это просто абзац</p>

Результат:



**Любой соседний элемент**

Для того, чтобы *выбрать* элемент, который находится после другого элемента, используется знак **~**. Пример:

img ~ p {

 margin-top: 0;

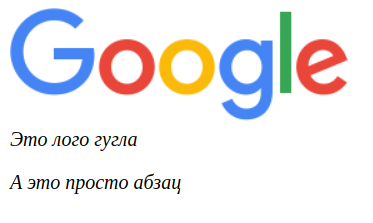
 font-style: italic;

}

Этот код применит стили во всех <p>, идущих **после** <img>, даже если между img и p есть другие элементы:

﻿

﻿Для HTML из предыдущего примера:



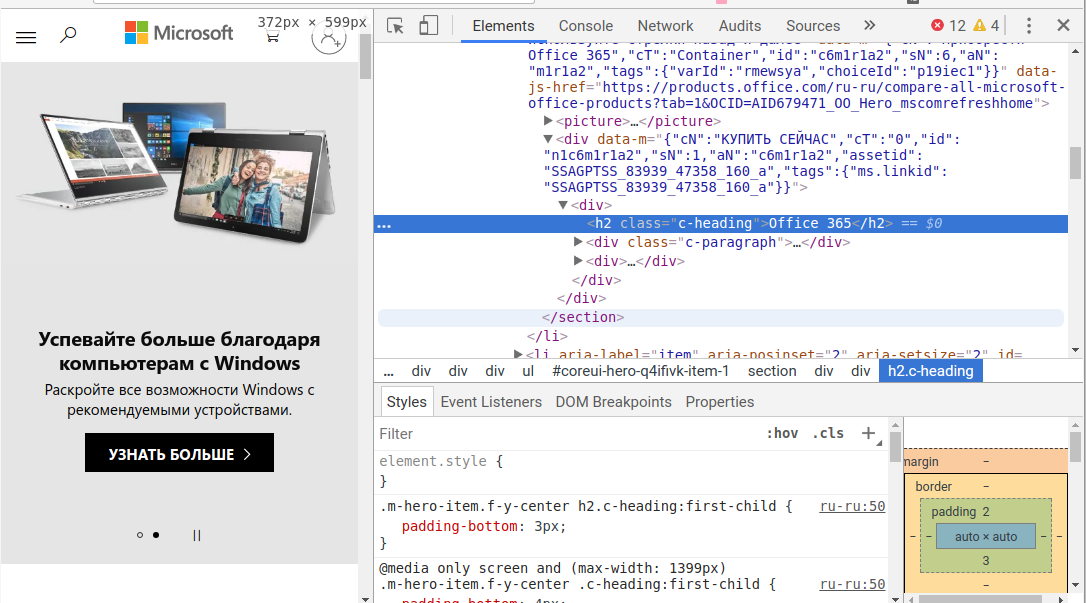
**Инструменты разработчика в браузере: Chromium DevTools**

Для разработки мы рекоммендуем использовать последний Chromium. Встроенные в него инструменты разработчика удобные и функциональные. В подавляющем большинстве обучающего материала в сети используются именно они.

Все браузеры, основанные на Chromium, подойдут: Google Chrome, Яндекс Браузер, Opera и даже Амиго.

**Взламываем Microsoft**

Зайди на [Microsoft.com](https://www.microsoft.com/ru-ru). Нажми на любой элемент правой кнопкой мыши и нажми "Inspect element" или «Показать код» или «Просмотреть код элемента». Откроется панель DevTools:



**Дерево элементов**. На этом курсе нам понадобится только вкладка "Elements". В ней отображаются узлы DOM-дерева (HTML-элементы) и их стили. При наведении на элемент из дерева, он будет подсвечен на странице. Можно увидеть контент, границы и отступы элемента. При двойном клике на содержимое элемента, можно его редактировать.

**Панель со стилями.**Мы будем использовать её, чтобы отлаживать наши домашние работы. В этой панели отображаются все CSS-свойства, которые применены к выбранному элементу, а так же его блочная модель.

**Поиграйся с этой штукой**: повыбирай разные элементы в дереве, попробуй менять и добавлять значения CSS-свойств в панели стилей.

**Текст и шрифт**

С помощью CSS можно делать любые операции с текстом и шрифтом. Самые основные действия это:

* **изменение свойств шрифта**: семейство, начертание, размер, толщина, стиль и т.д.
* **изменение свойств текста**: цвет, расстояние между буквами и строками, подчёркивание/надчёркивание/зачёркивание, выравнивание и т.д.

Чтобы узнать, как применить нужный стиль, просто загугли или воспользуйся справочником.

**Если во время выполнения задания, твой код работает не так, как нужно, попытайся найти причину через DevTools, перед тем, как задавать вопрос в комментарии или**[**учебный чат**](http://teleg.run/itc_html_css)**.**

**Семейства шрифтов и fallback**

Существует такое понятие, как семейство шрифтов. В HTML шрифты делятся на 4 базовых семейства:

**serif - шрифты с засечками или *антиква***

**

**sans-serif - шрифты без засечек, *гротеск* или *рубленые шрифты***

**

*﻿*

**monospace - моноширинные шрифты**



﻿

**cursive - рукописные шрифты**



﻿

**fantasy - аллегоричные шрифты - шрифты, которые не попадают в остальные семейства**



﻿

Когда мы используем определённый шрифт, мы указываем его имя в свойстве **font-family:**

p {

font-family: 'Roboto';

}

Но у такого подхода есть одна проблема: **если на устройстве пользователя нет этого шрифта, то он не увидит текста**.

Эта проблема решается с помощью fallback (фоллбэк):

p {

font-family: 'Roboto', 'Helvetica Nue', sans-serif;

}

Шрифты, которые идут через запятую после основного шрифта, называются *fallback fonts*(*фоллбэк-шрифты*). Обычно их несколько - следующий шрифт применяется, если не найден предыдущий.

**В конце списка фоллбэк-шрифтов всегда указывается семейство, в которое входят предыдущие шрифты.** Таким образом, если у пользователя нет конкретных, указанных нами шрифтов, то применится любой шрифт из указанного семейства, которой есть у него на устройстве.

Изображения шрифтов взяты отсюда: <http://www.quickandlazy.appspot.com/>

**Фон**

В CSS блокам можно задавать фон. Фоном может быть цвет или картинка. Картинку можно располагать с помощью координат, центрировать, растягивать на весь блок, или задавать в виде повторяющегося бесконечного паттерна. Всё это делается с помощью свойства **background,**подробнее про него здесь: <https://webref.ru/css/background>, обрати внимание на блоки «См. также» и «Рецепты».

Примеры:

.example {

/\* цвет фона «продающий красный», задан в HEX-формате \*/

background: #e64c0c;

}

.example {

/\* картинка на фоне \*/

background: url('images/nude-president.png');

/\* background-size отвечает за размер картинки, cover - картинка растягивается на весь блок, сохраняя пропорции \*/

background-size: cover;

}

**Цвет**

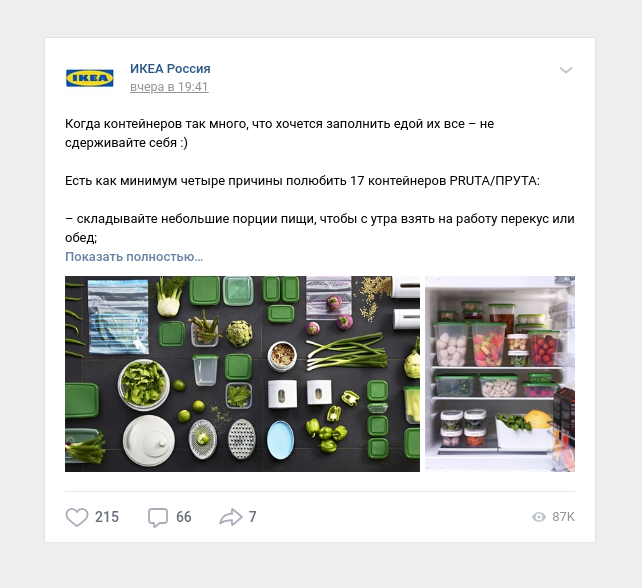
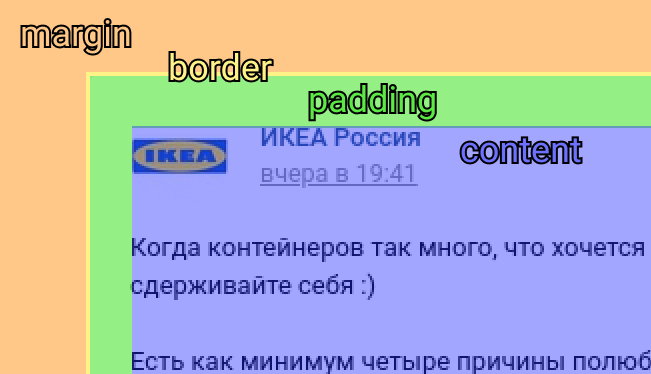
Цвет можно ставить не только фону, но и тексту, границам, теням. Есть несколько способов указать цвет:

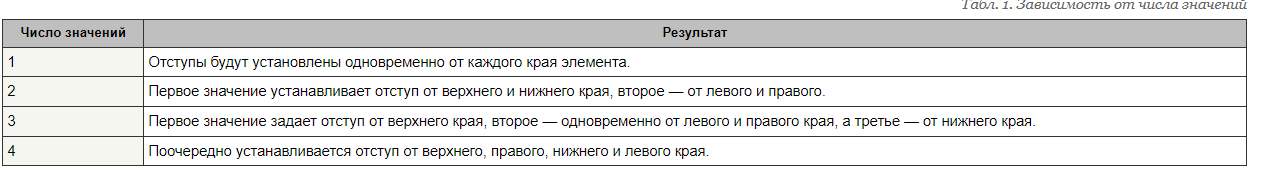
* **Имя цвета** из [списка HTML-цветов](https://htmlcolorcodes.com/color-names/), например "red", "green", "blue" , "

lemonchiffon".

* **HEX**, цвет в шестнадцатеричном формате. Начинается с решётки, например [#3ae6сa](https://vk.com/feed?section=search&q=%233aE6Ca). Можно подобрать с помощью [любого color picker'а](https://google.com/search?q=color+picker). Первые шесть символов отвечают за цвет, последние два - необязательные - отвечают за прозрачность. Но прозрачность удобнее задавать через RGBA и HSLA, потому что десятичные дроби удобнее читать и редактировать.﻿
* **RGB** (red, green, blue) - цвет, заданый тремя значениями (от 0 до 255) базовых цветов - красного, зелёного и синего. Пример: rbg(255, 123, 13).
* **RGBA** - тоже самое, что и RGB, только с прозрачностью, которая задаётся четвёртым числом от 0 до 1, пример: rgba(144, 32, 16, 0.5).
* **HSL, HSLA** - цветовая модель, в которой цветовыми координатами являются тон, насыщенность и светлота. [Подробнее о ней](http://igorshevchenko.ru/links/28). HSLA - HSL + прозрачность. Пример: hsl(120, 100%, 50%).
* **Тип отображения элемента (display)**
* В уроке «Элементы и их виды», мы узнали, что есть два типа элементов - блочные и строчные. Кратко: блочные элементы занимают всю доступную ширину и начинаются с новой строчки. Строчные занимают столько места по ширине, сколько занимает их содержимое, выстраиваются в одну строчку и переносятся, если им не хватает места.  
    
  С помощью CSS-свойства **display** можно переопределить тип отображения элемента. Оно может иметь одно из следующих значений.
* **display: none;** - элемент перестаёт отображаться на странице.
* **display: block;**- блочный элемент. Ему можно задать ширину, высоту, границы, отступы.
* **display: inline;**- строчный элемент. Задание ширины и высоты не влияет на inline элементы. Задание границ и отступов будет изменять положение окружающего текста, но не будет влиять на положение окружающих блочных элементов.
* **display: inline-block;** - что-то среднее между блочным и строчным элементом. Ему можно задать ширину, высоту, границы и отступы, но он не будет создавать перенос строки до и после себя, в отличие от блочных элементов. **С помощью этого типа можно распологать блоки горизонтально в ряд.**
* **flex и inline-flex**- это флексбоксы. Про них расскажем в уроке «Flexbox» в модуле «Продвинутая вёрстка». Элементы внутри них располагаются по определённым правилам, но снаружи они ведут себя как блоки и инлайн-блоки соотвественно.
* В дальнейшем элементы с display block, inline-block, flex и inline-flex мы будем называть просто **«блоки»**.

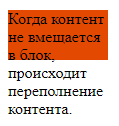
## На практике

* Возьмём карточку поста из Вконтакте.
* 
* Вот так выглядит её блочная модель:
* 
* **margin** - внешний отступ;  
  **border** - граница, между внутренним и внешним;  
  **padding** - внутренний отступ, под этим отступом продолжается фон элемента;  
  **content** - содержимое элемента;



# Переполнение контента

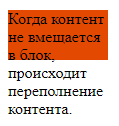
Когда контент не вмещается в блок, происходит переполнение контента и в блоке появляется полоса прокрутки. Пример: <https://codepen.io/mlshv/pen/ZmJgey>



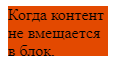
Свойство **overflow**управляет отображением содержания блочного элемента при переполнении.

## Значения overflow

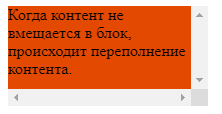
**overflow: visible;**- отображается все содержание блока, даже за пределами установленной высоты и ширины.



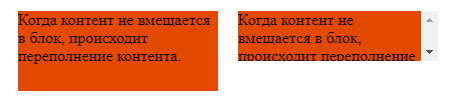
**overflow: hidden;** - отображается только область внутри блока, остальное будет скрыто.



**overflow: scroll;** - всегда добавляются полосы прокрутки, даже если контент помещается.



**overflow: auto;** - полосы прокрутки добавляются только при необходимости.



**Обтекание элементов (float)**

Если у тебя получилось сделать [второе домашнее задание](https://stepik.org/lesson/189513/step/1?unit=164012), то ты знаешь свойство **float**. Оно задаёт, по какому краю (левому или правому) будет выровнен элемент. С противоположной стороны его будут обтекать другие элементы - текст и блоки.

**Пример:**



*Картинка с фото со спутника обтекается текстом слева (****float: right****)*

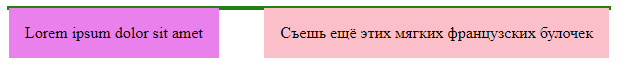
Свойство float может иметь следующие значения:

* left - выравнивание по левой стороне, обтекание справа,
* right  - выравнивание по правой стороне, обтекание слева,
* none - выравнивание не задаётся (нужно, чтобы сбросить ранее заданное значение).

# Clearfix

* При использовании свойства **float**, новички испытывают проблемы. И все они вызваны нелогичным принципом работы **float.**Рассмотрим несколько проблем и попытаемся их решить.

## Проблема 1

* Родительский блок, который содержит только элементы со свойством **float**, "схлопывается" и принимает высоту равную нулю.  
  Пример: [https://codepen.io/mlshv/pen/dQzxwE](https://codepen.io/mlshv/pen/dQzxwE?editors=1100).
* 

## Проблема 2

* Если в родительском блоке помимо "плавающих" (floating) элементов есть статические элементы, к которым мы не применяли **float**, то высота родителя станет равной высоте этих статических элементов. Пример: [https://codepen.io/mlshv/pen/aQyeMQ](https://codepen.io/mlshv/pen/aQyeMQ?editors=1100).
* 

## Проблема 3

* Следствие проблемы 2. Такие "вываливающиеся" блоки будут обтекаться контентом блоков, которые идут после родительского. Пример: [https://codepen.io/mlshv/pen/LXjwKo](https://codepen.io/mlshv/pen/LXjwKo?editors=1100).
* 

## Решение

* Чтобы избавиться от этих проблем, необходимо после "плавающих" блоков очищать обтекания, чтобы последующие элементы не стали обтекать "плавающие" блоки.
* Самое простое решение - очищающий блок. После плавающих блоков нужно добавить еще один блок и задать ему CSS:
* clear: both;
* Пример: [https://codepen.io/mlshv/pen/bQobdW](https://codepen.io/mlshv/pen/bQobdW?editors=1100)

Позиционирование

Позиционирование определяет, к какой части страницы элемент будет «прикреплён».

Тип позиционирования задаётся с помощью свойства position. Разберём типы позиционирования.

Статическое позиционирование (static)

Этот тип стоит у элементов по умолчанию. Элемент располагается в потоке относительно остальных элементов на странице. В потоке, то есть в таком порядке, что он идёт после элементов, которые объявлены перед ним. И элементы, которые объявлены после него, идут после него.

<h1>Заголовок</h1>

<p>Абзац</p>

Вроде бы очевидно, что если заголовок объявлен раньше абзаца, то он будет под заголовком. Да? Да, но только для этого типа позиционирования.

Абсолютное позиционирование (absolute)

Забавно, что абсолютное позиционирование - тоже относительное. Но задаётся оно по-другому.

Положение абсолютного блока задаётся относительно страницы или относительно элемента с position: relative, в которой он вложен. Положение такого блока задаётся с помощью свойств top, right, bottom, left.

Например, расположим абзац из примера выше вверху экрана. Для HTML кода из примера выше:

CSS:

p {

position: absolute;

top: 0;

}

Элементы со статическим позиционированием игнорируют абсолютные блоки, которые как бы "вываливаются" из потока и живут своей собственной жизнью. Как будто существуют на своём собственном слое. ﻿

Фиксированное позиционирование (fixed)

Задаётся так же, как и абсолютное, но блок прикрепляется не к странице, а к экрану. То есть при прокрутке страницы он остаётся на своём месте. Пример - шапка на степике.

Относительное позиционирование (position: relative)

Положение элемента устанавливается относительно его исходного места в потоке. Свойства left, top, right и bottom изменяет позицию элемента и сдвигает его в ту или иную сторону от первоначального расположения. А ещё оно работает как родитель, относительно которого можно задавать абсолютную позицию блокам, которые в него вложены.

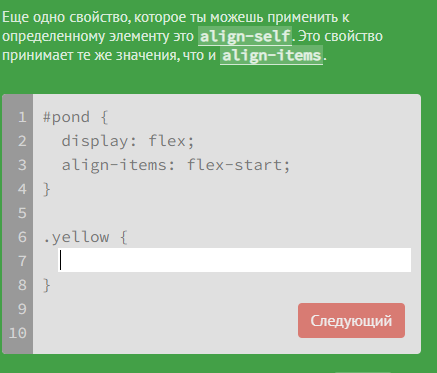
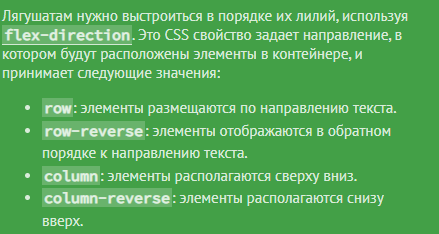
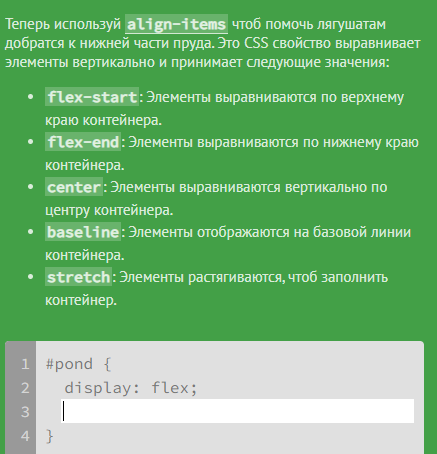
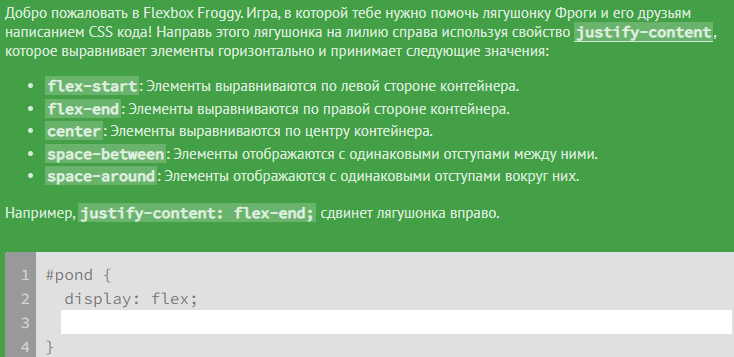
**Как изучать flexbox**

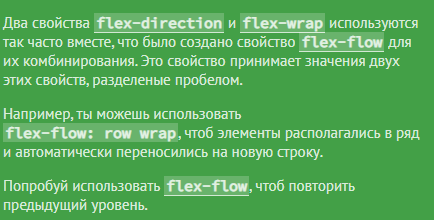
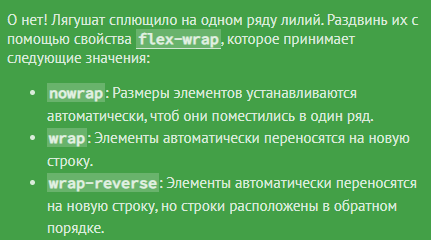
Перед тобой самая простая инструкция по изучению Flexbox.

1. Пройди игру [Flexbox Froggy](https://flexboxfroggy.com/" \l "ru" \o "Link: https://flexboxfroggy.com/#ru" \t "_blank)
2. Когда будешь писать код, держи под рукой вот этот гайд: <https://css-tricks.com/snippets/css/a-guide-to-flexbox/>

Всё! Теперь ты можешь использовать Flexbox в вёрстке своих проектов!

*Update:*﻿ученик [Tom Tom](https://stepik.org/users/30117234) предложил хорошую и интерактивную ﻿шпаргалку по флексам. Она правда очень крутая:  <https://tpverstak.ru/flex-cheatsheet/> ﻿





# Псевдоклассы

Псевдоклассы - это ключевые слова, которые добавляются к селектору. Стили, заданные таким образом, применяются только для определённых состояний. Например, **:hover** может быть использован для изменения цвета кнопки при наведении курсора:

button:hover {

background: red;

}

Стили псевдокласса переопределяют заданные ранее стили, если они есть. Пример:

button {

  border: 1px solid black;

  padding: 8px 16px;

  background: #c9c9c9;

}

button:hover {

  background: #e9e9e9;

}

Результат выполнения: [https://codepen.io/mlshv/pen/zyGdzq?editors=1100﻿](https://codepen.io/mlshv/pen/zyGdzq?editors=1100) (наведи мышь на кнопку)

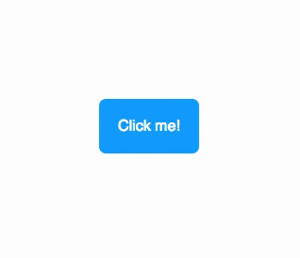
Самые часто используемые псевдоклассы:

* :hover - появляется при наведении мышки
* :active - появляется при нажатии на элемент
* :focus - появляется при фокусировке на элементе (например, когда выбрано поле ввода текста)

[Подробнее про псевдоклассы читай на MDN](https://developer.mozilla.org/ru/docs/Web/CSS/%D0%9F%D1%81%D0%B5%D0%B2%D0%B4%D0%BE-%D0%BA%D0%BB%D0%B0%D1%81%D1%81%D1%8B). Здесь же есть полный список псевдоклассов.

**Анимация переходов**

Переходы состояний некоторых в CSS можно анимировать. Это работает так: если значение свойства меняется, то вместо резкого изменения, происходит плавный переход. Пример:



Вот так выглядит анимирование переходов в коде:

button {

border-radius: 4px;

transition: border-radius 0.5s ease-in;

}

button:hover {

border-radius: 16px;

}

За анимацию отвечает свойство **transition**. Его значние записывается так:

1. свойство, изменения которого нужно анимировать (если нужно анимировать несколько, можно написать **all**)
2. продолжительность анимации в секундах (рекомендуется не ставить её больше 0.3 секунды)
3. функция плавности, подробнее здесь: <https://easings.net/ru>

Вот полный список доступных для анимаций свойств: <https://developer.mozilla.org/en-US/docs/Web/CSS/CSS_animated_properties>

**Медиа-запросы**

Медиа-запросы позволяют задавать стили для определенных типов устройств, например, принтеры, экраны или синтезаторы речи.

В примерах мы будем рассматривать медиа-запросы для экранов различной ширины. Остальные типы используются редко, просто знай, что они есть, чтобы загуглить, когда понадобится.

Чаще всего они используются, чтобы создавать адаптивные страницы.

**Синтаксис**

Синтаксис медиа-запроса выглядит так:

﻿@media правило {

/\* ﻿обычный CSS-код, который применится при соблюдении правила \*/

}

﻿Пример медиа-запроса:

@media screen and (max-width: 320px) {

img {

width: 100%;

}

}

Этот код означает следующее: **на экранах**, **ширина** которых **меньше или равна 320px,**картинки будут иметь ширину 100%.

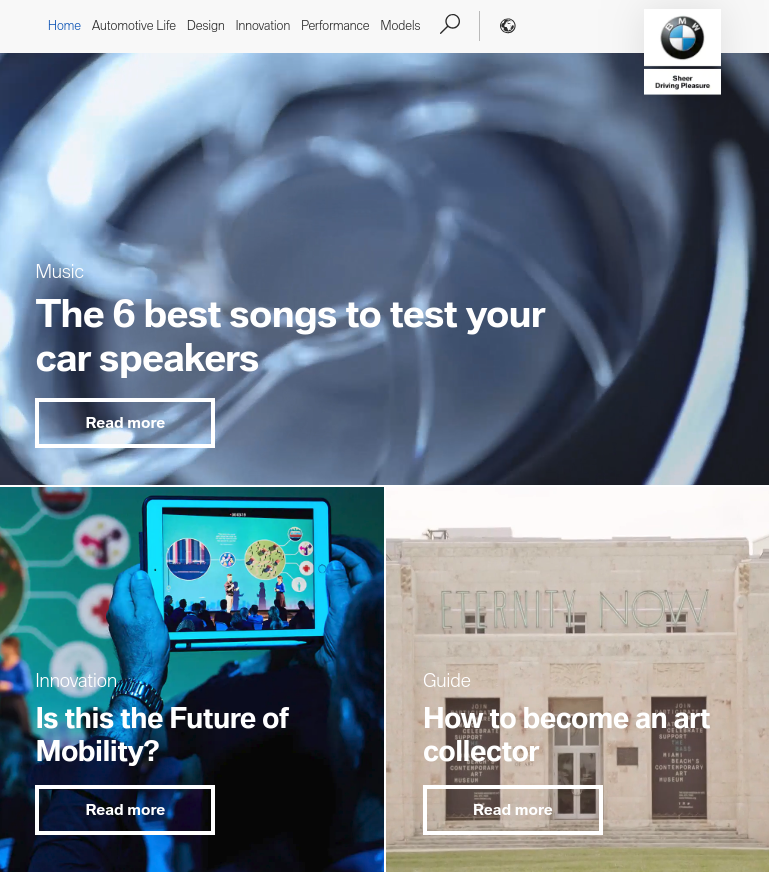
Подробнее про синтаксис читай в документации MDN: <https://developer.mozilla.org/ru/docs/Web/CSS/Media_Queries/Using_media_queries>

**Пример**

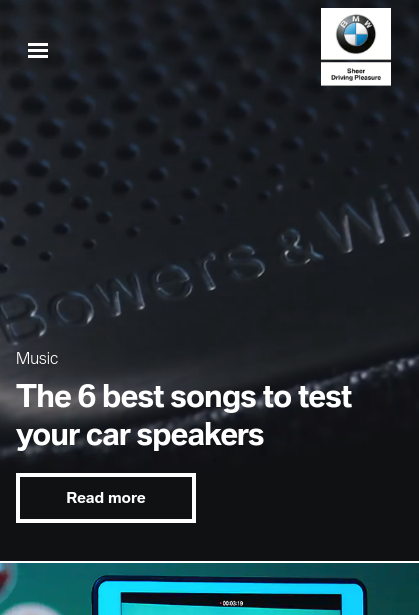
Открой вот эту страницу и попробуй уменьшить ширину окна: <https://www.bmw.com/en/index.html>.

Обрати внимание на блоки, которые выстраиваются в одну колонку и на верхнее меню, которое прячется под кнопку.

Скриншоты на случай, если ты сидишь с планшета или телефона:



*Версия для больших экранов*



*Версия для маленьких экранов*

**Медиа-запросы: практика 2**

Теперь макет попроще, а задание посложнее.

У нас есть страница, которая состоит из блоков. Схематично они представлены в виде серых прямоугольников. Адаптируй страницу под разные экраны:

**Узкие экраны (мобильные, ширина ﻿<992px)**

**.c6, .c2, .c1**: ширина 100%.

**Средние экраны (планшеты, ширина ≥992px)**

**.c6**: ширина 100%.  
**.c2**: ширина 50%.  
**.c1**: ширина 33.3333%. ﻿

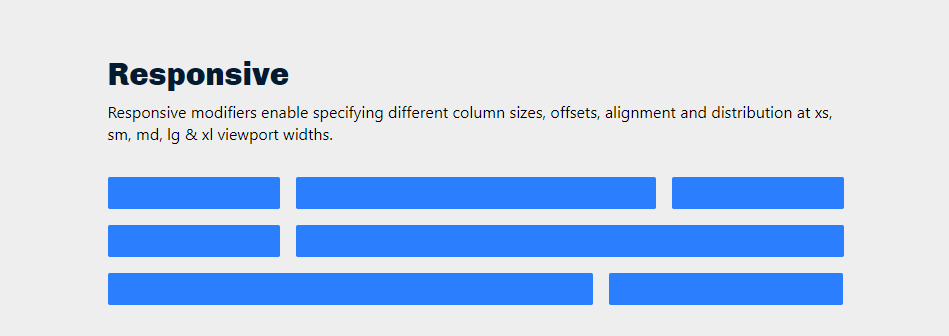
**Широкие экраны (настольные ПК, ширина ﻿ ≥1200px)**

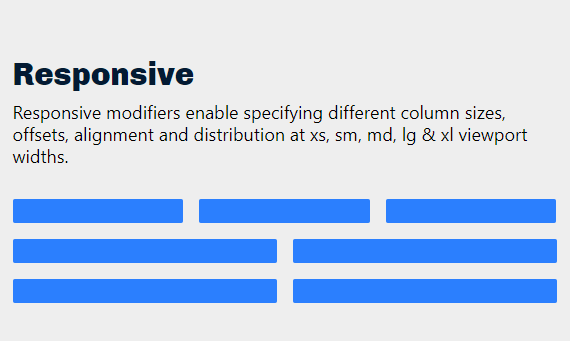
**.c6**: ширина 100%.  
**.c2**: ширина 33.3333%.  
**.c1**: ширина 16.6666%. ﻿

**Адаптивная сетка**

*Перед началом этого урока рекомендуется пройти*[*практику из предыдущего.*](https://stepik.org/lesson/181717/step/3?unit=156325)*Прохождение этого урока на планшетах и смартфонах - плохая идея.*  
  
Адаптивная сетка - структура, в которой размещаются блоки страницы. Сетка позволяет легко указывать, какое пространство блоки занимают на разных экранах.  
  
В прошлом уроке мы вручную расписали размер блоков для разных экранов через медиа-запросы. При использовании сетки этого делать не нужно. Всё, что от нас требуется - задать нужные классы в HTML для колонок.

Адаптивная сетка подключается как отдельный CSS-файл. В этом файле заданы стили для структурных элементов сетки: контейнер сетки, строка, колонки. Существует большое количество сеток, созданных различными разработчиками. Одна из них - flexboxgrid2. Открой страницу с демо и примерами и попробуй сузить окно браузера. Обрати внимание на поведение колонок: <https://evgenyrodionov.github.io/flexboxgrid2/>



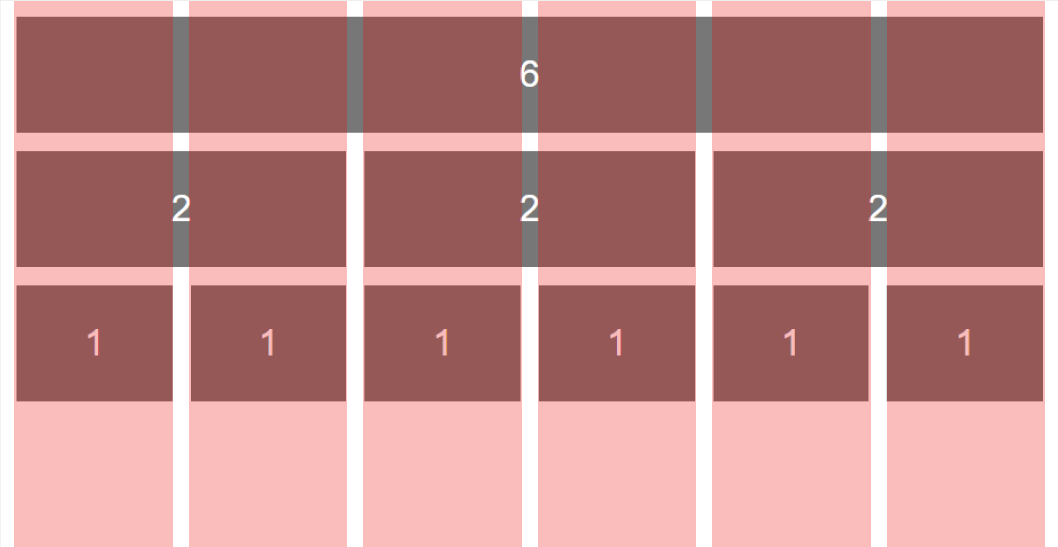




**Адаптивная сетка: колонки**

Самая главная вещь, которую нужно понять перед использованием адаптивной сетки, это колонки.

Любая сетка делится на колонки. На макете они выглядят как вертикальные полосы, по которым выровнены блоки страницы. Если нарисовать сетку поверх макета для задания «[Медиа-запросы: практика 2](https://stepik.org/lesson/181717/step/3?unit=156325)», то получится вот это:

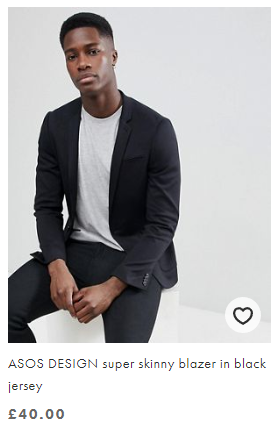


Эта сетка состоит из 6-ти колонок, но обычно сетки состоят из 12-ти: это число делится на 2, на 3, на 4 и на 6.

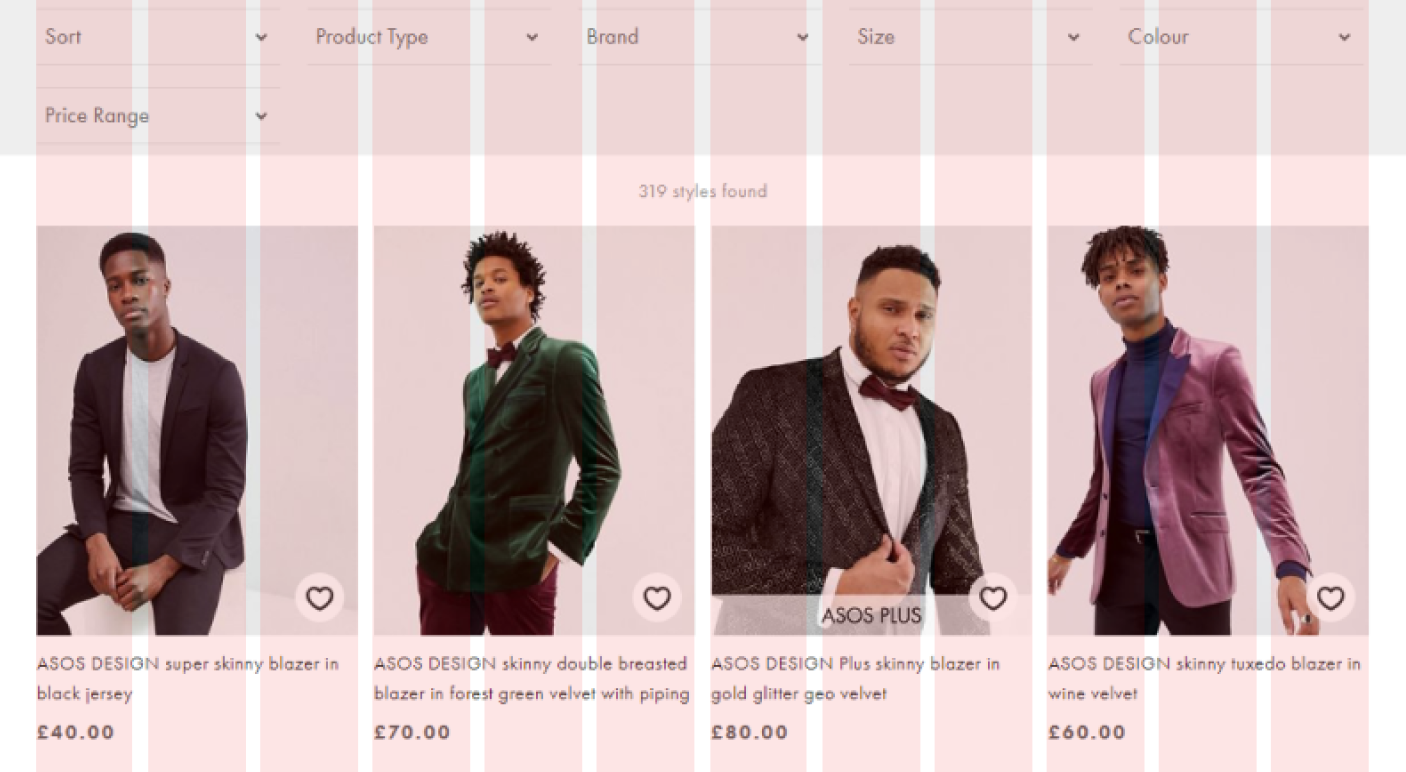
Удобство колонок в том, что**для одного и того же блока можно выбрать, сколько колонок он займёт на разных экранах.**

**Пример**

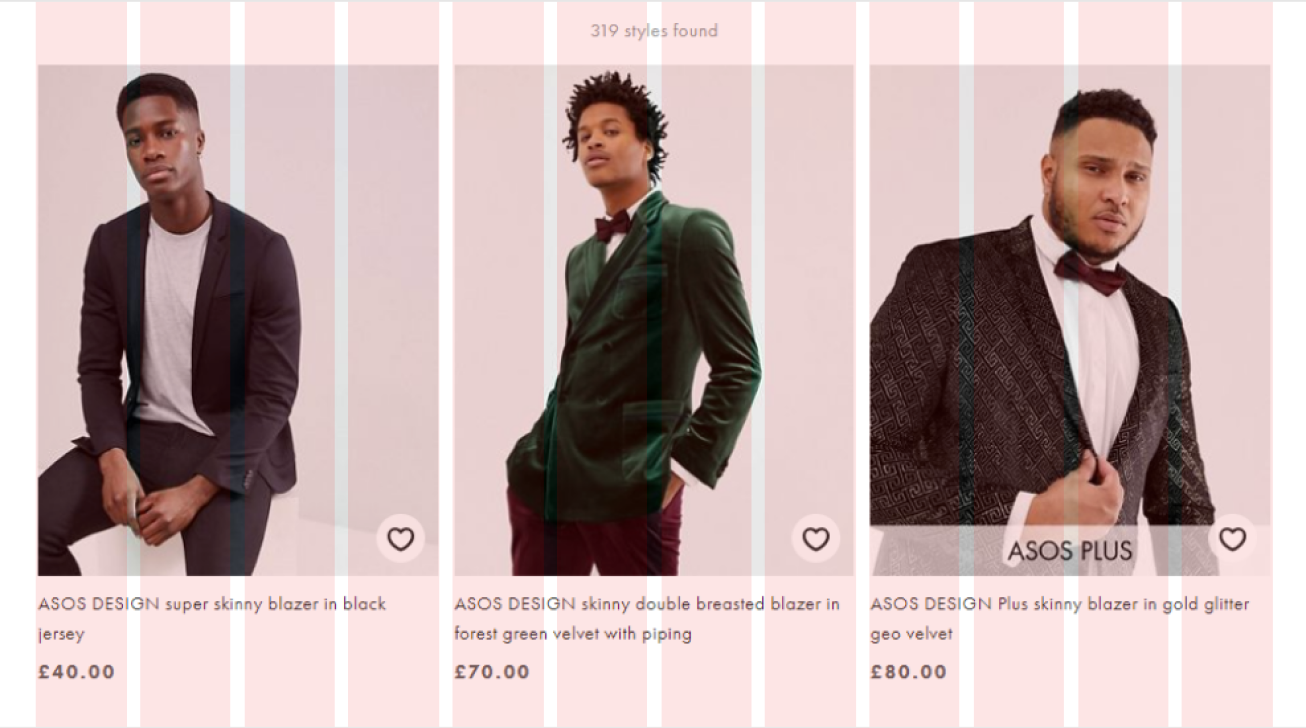
У нас есть блок - карточка товара:

****

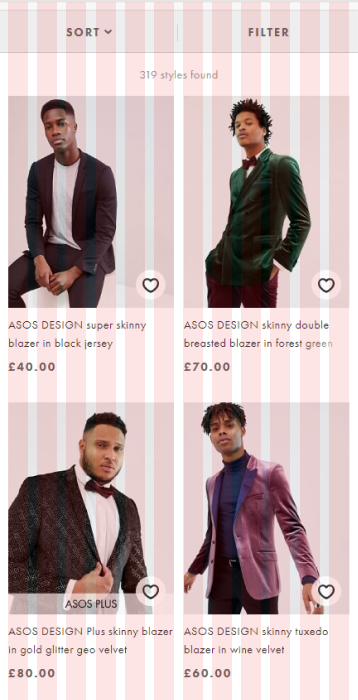
На большом экране эта карточка будет занимать 3 колонки:



На среднем - 4:



А на маленьком - 6:

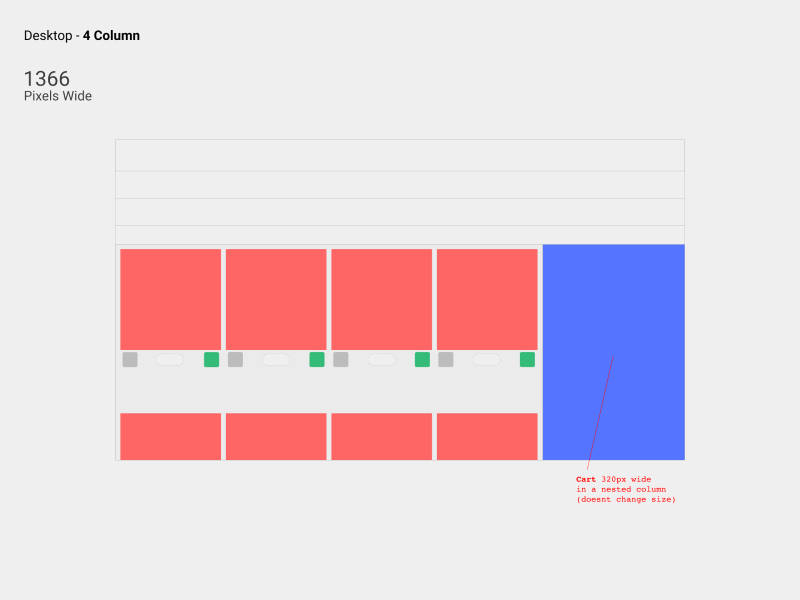


То есть **для каждого блока мы задаём, сколько колонок он займёт на нужных экранах**.

**Адаптивная сетка: брейкпоинты**

Breakpoints (*брейкпоинтс*) - значения ширины экрана, при которых происходит переключение отображения колонок. Проще говоря, это ширина, которая прописана в медиа-запросах сетки.

Если ты будешь менять ширину окна на странице, где есть сетка, то увидишь «щелчки» переключения отображения. Они происходят при переходе через брейкпоинты.



В разных сетках эти значения могут быть заданы по-разному. В flexboxgrid2 они вот такие:

* xs: 0..575px
* sm: 576..767px
* md: 768..991px
* lg: 992..1199px
* xl: 1200px+

# Каждая ширина экрана называется по-своему: *xs, sm, md, lg, xl*. От слов *extra small, small, medium, large, extra large*. Эти названия используются в именах классов. Адаптивная сетка: код

Мы будем пользоваться [flexboxgrid2](https://github.com/evgenyrodionov/flexboxgrid2). Не волнуйся, если ты планируешь пользоваться другой сеткой - принцип работы у всех них одинаковый.

**Пример #1**

Полностью аналогичен примеру из шага с карточками товара. Блок, который занимает 6 колонок на *xs,*4 колонки начиная с *md,*и 3 колонки с lg и выше.

<div class="container">

<div class="row">

<div class="col-xs-6 col-md-4 col-lg-3">

<div>Блок</div>

</div>

<div class="col-xs-6 col-md-4 col-lg-3">

<div>Блок</div>

</div>

<div class="col-xs-6 col-md-4 col-lg-3">

<div>Блок</div>

</div>

<div class="col-xs-6 col-md-4 col-lg-3">

<div>Блок</div>

</div>

</div>

</div>

Codepen: <https://codepen.io/mlshv/pen/yGJaoK?editors=1100> (попробуй менять ширину окна)

Обрати внимание на структуру:

* container - контейнер сетки
* row - ряд, строка сетки
* col-\*\*-\* - колонки

В именах колонок сначала идёт префикс col-, потом название экрана, для которого применяется значение и в конце - количество колонок, которые займёт блок.

**Пример #2**

У экранов есть приоритет. Если мы пропустим какой-то из экранов, и не зададим для него ширину в колонках, то автоматически будет использоваться ширина более узкого экрана. Если же у самого узкого экрана не будет значения, то оно будет считаться равным 12 колонкам. Пример:

<div class="container">

  <div class="row">

    <div class="col-lg-6">

<div>Блок</div>

    </div>

    <div class="col-lg-6">

<div>Блок</div>

    </div>

    <div class="col-lg-6">

<div>Блок</div>

    </div>

    <div class="col-lg-6">

<div>Блок</div>

    </div>

  </div>

</div>

Codepen: <https://codepen.io/mlshv/pen/jXrMaY?editors=1100>

Попробуй менять ширину окна. Хоть у блоков задана ширина только на одном экране (lg), на экранах шире lg будет применена та же ширина. А на более узких экранах - ширина по умолчанию - 12 колонок.

**Пример #3 (с ошибкой)**

Старайся не смешивать свои стили и стили сетки. Не пиши контейнеру, строке, колонке свои классы и стили. Это считается дурным тоном и снижает возможность переиспользования кода.

<div class="container">

  <div class="row">

    <div class="col-lg-6 block" style="background: #CC4A14">

      Блок

    </div>

    <div class="col-lg-6 block" style="background: #FF0000">

      Блок

    </div>

    <div class="col-lg-6 block" style="background: #FF00f0">

      Блок

    </div>

    <div class="col-lg-6 block" style="background: #40FF40">

      Блок

    </div>

  </div>

</div>

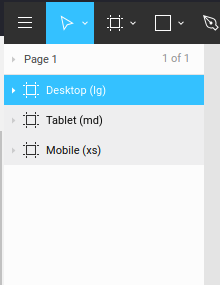
Не делай так!

**Адаптивная сетка: практика**

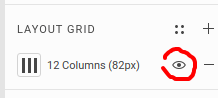
Задание на тренировку умения верстать колонки с макета.

Макет из фигмы: [https://www.figma.com/file/Ey4hA2woPeTiXiMnlKAYRF/Grid-Exercise](https://www.figma.com/file/Ey4hA2woPeTiXiMnlKAYRF/Grid-Exercise?node-id=0%3A1)

На нём отображены блоки, которые необходимо распределить по колонкам. ﻿Чтобы включить отображение сетки на макете, выбери нужный фрейм в меню слева:  
﻿



﻿  
и затем включи отображение сетки в меню справа: либо ctrl + shift + 4  
﻿



﻿  
Код: <https://codepen.io/mlshv/pen/Bvzoej?editors=1100>﻿ (первые два блока уже распределены)

# SS-фреймворки

По сути, CSS-фреймворк - это CSS-код, который написали одни разработчики и выложили его для других. Они нужны для того, чтобы экономить время при разработке, не изобретать свой велосипед без надобности. Обычно такие фреймворки включают в себя набор компонентов и адаптивную сетку.

﻿Чтобы использовать CSS-фреймворк, нужно подключить его файлы к HTML-странице. После этого надо ﻿просто добавлять классы HTML-элементам﻿, которые прописаны в файлах фреймворка, чтобы всё стало красиво.

Пример:

<button class="btn btn-primary">Primary</button>

﻿Результат:



Самый популярный CSS-фреймворк - Bootstrap от разработчиков Twitter. В этом модуле мы будем изучать именно его. Весь модуль является практическим - на следующем уроке мы создадим проект и будем выполнять в нём остальные задания.

Компоненты Bootstrap: <https://getbootstrap.com/docs/4.1/components>.

Примеры страниц на Bootstrap: <https://getbootstrap.com/docs/4.1/examples/>.

**Когда применять и не применять CSS-фреймворки**

**Плюсы фреймворков**

* Набор готовых UI-компонентов.
  + Скорость разработки выше.
  + Более красивый интерфейс, если у разработчиков плохой вкус или время на разработку ограничено.
* Многие разработчики с ними знакомы.
  + Проще поддерживать код.
  + Новым членам команды проще начать работу.
* Сообщество.
  + Много примеров использования и обучающих материалов.
  + Маленькое количество багов.
  + Легко нагуглить решение почти любой проблемы.

**Минусы фреймворков**

* Обязательство работать в рамках, заданных разработчиками фреймворка.
  + Если возникает проблема, решение которой не предусмотрено во фреймворке, приходится изворачиваться и делать «финты ушами».
  + Чем дальше развивается проект, тем сложнее поменять или избавиться от фреймворка.
* Обычно используется только часть возможностей фреймворка.
  + Неиспользуемый код увеличивает размер страниц, они загружаются медленее.
* Необходимо тратить время на изучение.
* Шаблонность: сайты, созданные на основе фреймворка обычно не отличаются индивидуальностью.

**Когда применять и не применять**

Фреймворки нужно использовать тогда, когда:

* время на разработку ограничено, и нужно запустить сайт как можно скорее,
* индивидуальность оформления и скорость загрузки сайта не так важны,
* проект не планируется сильно развивать в дальнейшем, a.k.a «сделал и забыл»,
* нет дизайнера и разработчик не может сам сделать красивый интерфейс.

## Создание проекта

Создадим новый проект — папку, в которой будет *index.html* и папка “*vendor”.*

*Названия “vendor” или “lib” носят папки, в которых лежит код третьей стороны (third-party code), то есть код, который используется в проекте, но не был написан его разработчиками. Обычно это библиотеки или файлы фреймворка.*

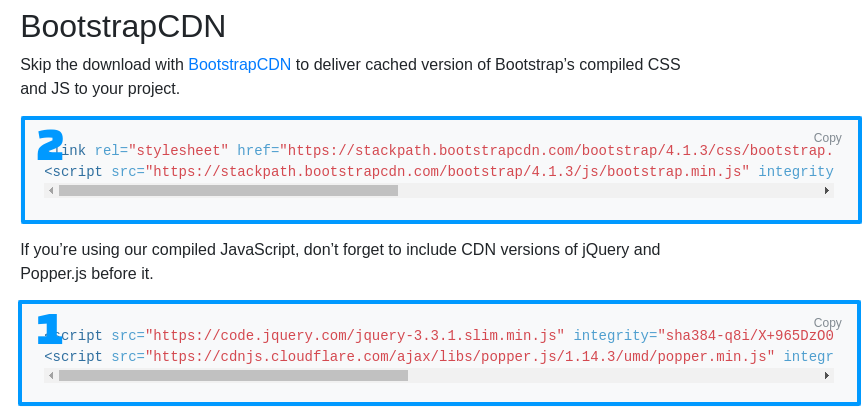
**Подключение Бутстрапа**

Сначала [заходим на сайт Бутстрапа](https://getbootstrap.com/docs/4.1/getting-started/download/). Там есть несколько способов подключения фреймворка, нам подходят два из них.

**Через CDN**

﻿Файлы Бутстрапа загружены на CDN. Это значит, что их можно не скачивать себе, а сразу подключить на страницу.

Копируем код из поля 1, потом из поля 2 в *<head>* документа. Удаляем папку vendor, так как в этом случае она не нужна.



**Локально**

﻿Нажимаем на кнопку “Download” у пункта “Compiled CSS and JS”.



Достаём из архива папки *css* и *js* и кладём их в *vendor*.

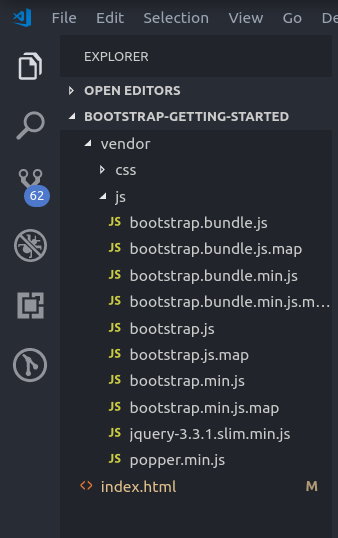
Скачиваем зависимости Бутстрапа: jquery и popper.js, для этого сначала переходим по ссылкам:

<https://code.jquery.com/jquery-3.3.1.slim.min.js>

<https://cdnjs.cloudflare.com/ajax/libs/popper.js/1.14.3/umd/popper.min.js>

После открытыия нажимаем Ctrl + S (или Cmd + S на маке) ﻿и сохраняем файлы в *vendor/js*.

Структура проекта теперь выглядит так:



Подключаем файлы в проект, добавляем вот этот код в *<head>*:

<script src="vendor/js/jquery-3.3.1.slim.min.js" defer></script>

<script src="vendor/js/popper.min.js" async></script>

<link rel="stylesheet" href="vendor/css/bootstrap.min.css">

<script src="vendor/js/bootstrap.min.js" defer></script>

**Проверка**

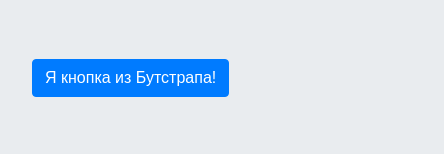
Проверим, что всё подключилось корректно,  вставив этот код в *<body>:*

<div class="jumbotron">

 <button class="btn btn-primary">Я кнопка из Бутстрапа!</button>

</div>

﻿Если на странице появилось синяя кнопка на сером фоне, значит всё работает правильно.  
﻿



**Сайт Бутстрапа**

Краткий обзор того, что полезного можно найти на сайте Бутстрапа.

**Документация**

Документация свежего Бутстрапа доступна по ссылке: <https://getbootstrap.com/docs/>

Тут можно прочитать всё про подключение и использование фреймворка, например:

* Быстрый старт и начальный шаблон страницы: <https://getbootstrap.com/docs/4.2/getting-started/introduction/>
* Описание всех компонентов и примеры к ним (про компоненты написано в следующем уроке)
* Рекомендации по использованию иконок: <https://getbootstrap.com/docs/4.2/extend/icons/>
* Утилиты для вёрстки: <https://getbootstrap.com/docs/4.2/utilities/>
* Гайды по оформлению кода на веб-странице: <https://getbootstrap.com/docs/4.2/content/code/>
* ... и многое другое

**Примеры**

Официальные примеры страниц на Бутстрапе. Многие из них используют немного дополнительного CSS, который не входит во фреймворк. Для изучения примеров используй DevTools.

<https://getbootstrap.com/docs/4.2/examples/>

**Темы**

Темы, которые можно подключить к Бутстрапу, чтобы никто не понял, что это Бутстрап: <https://themes.getbootstrap.com/>﻿

**Expo**

Красивые сайты, которые были созданы с помощью фреймворка. Можно использовать для вдохновения, обучения или *воровства* *идей*.

<https://expo.getbootstrap.com/>

**﻿Bootstrap: компоненты**

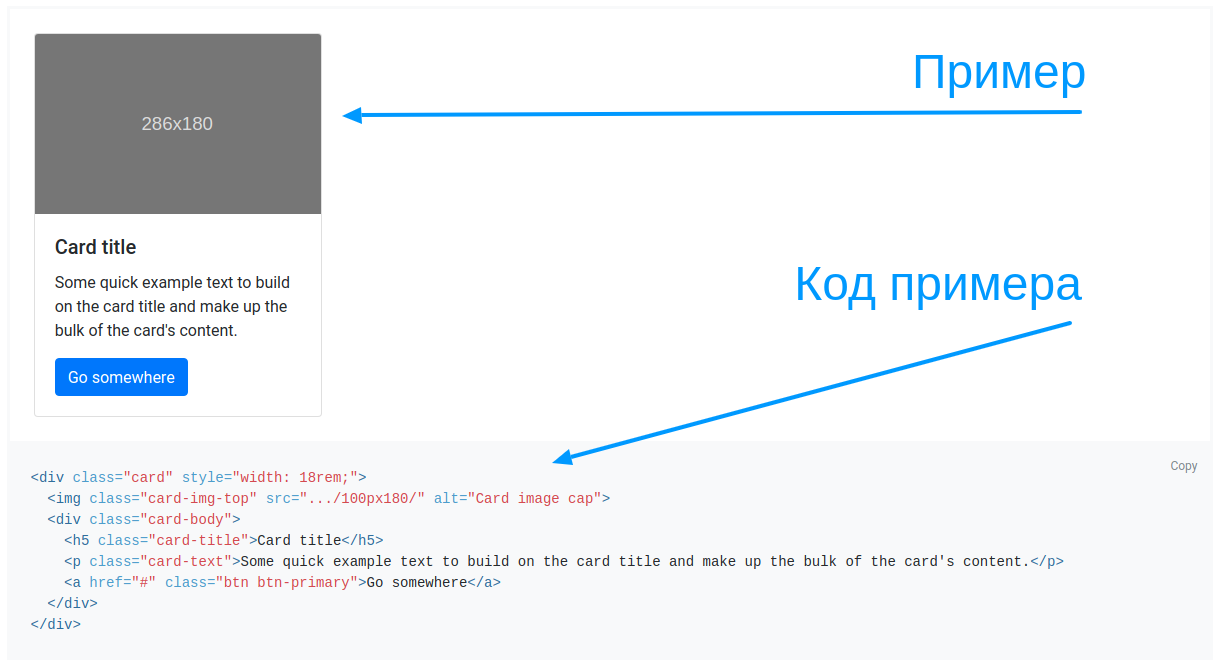
В Bootstrap есть много готовых компонентов, которые упрощают жизнь разработчика и пользователя. Кнопки, переключатели, карточки, элементы навигации и многое другое.

Полный список компонентов: <https://getbootstrap.com/docs/4.1/components>

**Как пользоваться компонентами Bootstrap**

На странице каждого компонента есть пример с кодом, в котором он используется. Если компонент достаточно сложный, то у него есть несколько вариаций. Они представлены в виде отедельных примеров-рецептов.

Например, карточки: <https://getbootstrap.com/docs/4.1/components/card/>



Чтобы использовать компонент, тебе достаточно скопировать код и вставить к себе на страницу (не забудь подключить сам Bootstrap)

**﻿Bootstrap: сетка**

В Bootstrap встроена адаптивная сетка. Она является самой популярной и известной, хотя и имеет свои минусы.

Страница сетки на сайте Bootstrap: <https://getbootstrap.com/docs/4.1/layout/grid/>﻿

Сетка делится на стандартные 12 колонок и имеет следующие брейкпоинты.

**Брейкпоинты**

**Экстра-маленький**<576px  
Названия колонок: *col-xs-\**

**Маленький**≥576px  
Названия колонок:*col-sm-\**

**Средний** ≥768px  
Названия колонок:*col-md-\**

**Большой** ≥992px  
Названия колонок:*col-lg-\**

**Экстра-большой**﻿≥1200px  
Названия колонок:*col-xl-\**

Звёздочкой в названии колонок обозначается число колонок, которые займёт блок, см. [«Адаптивная сетка»](https://stepik.org/lesson/197141/?unit=171379).

**Макет сетки**

То, как выглядит сетка, можно посмотреть на этом макете в Фигме:  
[﻿https://www.figma.com/file/Ymijl6oFHyJWBKt67eKaRMDz/Bootstrap-Grid](https://www.figma.com/file/Ymijl6oFHyJWBKt67eKaRMDz/Bootstrap-Grid)﻿.

﻿Можешь использовать её при создании макетов или отправить своему ﻿дизайнеру.