# [Знакомство](https://htmlacademy.ru/courses/4)

## Одиночные HTML-теги [5/14]

С некоторыми [парными тегами](https://htmlacademy.ru/courses/4/run/4) мы познакомились. А что же за одиночные теги?

Парные теги обычно нужны, чтобы оформить некоторый участок текста. Благодаря паре тегов вы можете указать начало и конец этого участка. Но ведь есть теги, которые не предназначены для оформления фрагментов текста.

Например, тег для вставки изображения или тег для вставки разделительной полосы. Такие теги добавляют на страницу одиночный объект, и им не нужно для этого заключать в себя какой-то текст. Поэтому их называют одиночными.

Примеры таких тегов: <br>, <hr>, <img>.

Кстати, в HTML-редакторе вы увидите такие фрагменты кода: <!-- текст -->. Они называются «комментарии», и браузер не отображает их на странице.

Раньше одиночные теги писались с закрывающим слешом перед закрывающей скобкой. Например: <br/>.  
В новом стандарте HTML5 использование закрывающего слеша в одиночных тегах необязательно.

## Атрибуты HTML-тегов [6/14]

[В предыдущем задании](https://htmlacademy.ru/courses/4/run/5), как вы заметили, после вставки в код тега <img> ничего не произошло. Почему же так вышло?

Теги могут иметь атрибуты. Некоторые теги есть смысл использовать только с атрибутами. Наиболее яркий пример — тег <img>, обозначающий изображение. Для него обязательно нужно указывать атрибут src, который задаёт адрес картинки (иначе браузер не сможет загрузить её).

В общем случае тег записывается следующим образом:

<имя-тега атрибут1="значение1" атрибут2="значение2" ...>

Атрибутов может быть несколько, вот примеры:

<p class="important">...</p>

<a class="external" href="https://htmlacademy.ru">...</a>

<img class="avatar" src="keks.png">

В этом задании вы потренируетесь использовать атрибуты тегов.

Не забудьте пробелы между названием тега и атрибутом и между атрибутами

## Азы CSS [8/14]

CSS — это язык для управления внешним видом HTML-документа. С помощью CSS можно задавать параметры отображения любого тега: ширину и высоту, отступы, цвет и размер шрифта, фон и так далее.

CSS это аббревиатура «Cascading Style Sheets» или «Каскадные Таблицы Стилей». Обычно CSS называют просто «стили».

Самый простой способ применить стили к тегу заключается в использовании атрибута style. Например:

<p style="color: red;">...</p>

В этом примере абзацу задан красный цвет шрифта. Такой способ задания стилей называют «инлайновые стили» или «встроенные стили».

Синтаксис таких стилей очень простой: свойство: значение;. Причём свойств может быть несколько.

Теперь давайте составим мини-конспект курса и отметим зелёным цветом те темы, которые мы уже отработали на практике. Для стилизации используем инлайновые стили.

## Другие способы подключения CSS [9/14]

Задавать стили каждого тега с помощью атрибута style очень затратно и хлопотно. А ещё это приводит к засорению HTML-кода избыточными, повторяющимися кусками CSS.

К счастью, есть и другие способы подключения стилей. Первый — подключение внешнего файла с помощью тега <link>, а второй — использование специального тега <style>. Подробнее эти методы будут разобраны в последующих курсах.

А сейчас вы познакомитесь со вторым разделом редактора, помеченным как CSS. Код из CSS-редактора подставляется в HTML-документ так, как будто вы записали его в тег <style>.

Для начала мы почистим код в HTML-редакторе, а потом начнём пошагово стилизовать наш мини-конспект с помощью других возможностей CSS.

## Селекторы в CSS [10/14]

Поздравляем! Вы только что [подключили внешние CSS-стили](https://htmlacademy.ru/courses/4/run/9) к своей HTML-страничке.

Когда вы задаёте стили тега с помощью атрибута style, браузер сразу же понимает, к какому именно тегу применить эти стили. Но когда стили подключаются с помощью внешнего файла или через тег <style>, браузер ищет стилизуемые теги с помощью «селекторов».

С селекторами вы уже немного знакомы: в предыдущем задании вы использовали селектор p, который находился перед фигурными скобочками в CSS-коде. В общем случае синтаксис CSS-правил выглядит так:

селектор {

свойство1: значение1;

свойство2: значение2;

...

}

Язык селекторов очень мощный и гибкий. Простейший тип селекторов — селекторы по имени тега: p, h1 и так далее. Когда браузер видит такой селектор, он применяет стили из правила ко всем подходящим тегам. Например, ко всем абзацам или ко всем заголовкам первого уровня.

Но, Хьюстон, у нас проблемы с конспектом! Селектор p подсветил все абзацы зелёным. А они должны быть разного цвета. Как быть?

Давайте попробуем добавить внутрь абзацев разные теги для разных цветов. Тогда в CSS можно будет использовать разные селекторы.

## Классы в CSS [11/14]

Что ж, [конспект мы снова раскрасили](https://htmlacademy.ru/courses/4/run/10), но какой ценой. Давайте признаем, что добавлять внутрь абзацев дополнительные теги и раскрашивать текст с их помощью, не самое лучшее решение. Ну а какое решение хорошее?

Такое решение есть и оно называется «классы»!

Вспомните шаг, когда мы пытались стилизовать абзацы с помощью внешних стилей, но все они получились одинакового цвета.

Нужно было сделать следующее: добавить абзацам разные классы, а в CSS использовать селекторы не по тегу, а по классам.

Класс — это всего лишь один из атрибутов HTML-тегов, например:

<p class="important">...</p>

<p class="help">...</p>

В CSS можно задавать стили только для элементов с определённым классом. Для этого используется селектор по классу, который пишется так .имя-класса, например:

.important { color: red; } — выберет все теги с классом "important"

.help { color: green; } — выберет все теги с классом "help"

Классы гибкие, их можно создавать много и называть понятными именами. Например, можно создать класс, который отмечает раздел курса, который сейчас изучается.

Имя класса может содержать в себе латинские символы, цифры, символ дефиса -и подчёркивания \_ и начинаться оно должно с латинского символа.

## Свойства и значения CSS [12/14]

Итак, язык CSS состоит из [селекторов](https://htmlacademy.ru/courses/4/run/10) и свойств. Селекторы указывают на то, к каким элементам применять стили, а свойства — на то, как именно отображать элементы.

Существует огромное количество CSS-свойств, которые влияют практически на все аспекты отображения элементов. Причём каждому свойству соответствует определённый набор значений.

Некоторые значения задаются с помощью текстовых констант, например red, bold, другие с помощью цифровых значений: 12px, 120% и так далее.

Мощь стилей заключается в том, что вы можете быстро и гибко менять внешний вид нужных элементов, особенно когда используете классы. Например, чтобы зачеркнуть текст всех изученных пунктов конспекта, нужно добавить всего лишь одну строчку в CSS:

.learned-ok {

color: green;

**text-decoration: line-through;**

}

И все теги с классом .learned-ok будут отображаться с перечёркнутым текстом. Теперь представьте, как долго делать то же самое через инлайновые стили, когда в конспекте больше сотни пунктов.

# Структура HTML-документа

## С чего начинается HTML

Каждый HTML-документ должен начинаться с декларации типа документа или «доктайпа». Тип документа нужен, чтобы браузер мог определить версию HTML и правильно отобразить страницу.

Например, для старой версии HTML 4.01 доктайп выглядит так:

<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"

"http://www.w3.org/TR/html4/strict.dtd">

А для последней версии HTML уже намного проще:

<!DOCTYPE html>

Последнюю версию HTML ещё называют HTML 5. Но так как эта версия уже принята как стандарт и распространена почти везде, мы будем называть её просто HTML.

## Ключевые слова

Есть целое семейство тегов <meta>, называемых мета-тегами. Их можно использовать внутри тега <head>.

Мета-теги различаются набором атрибутов и их значений, вот некоторые из атрибутов: content, http-equiv, name и scheme.

Мета-теги хранят полезную для браузеров и поисковых систем информацию. Один из таких тегов — это описание ключевых слов страницы. Задаётся он так:

<meta name="keywords" content="разные, ключевые, слова">

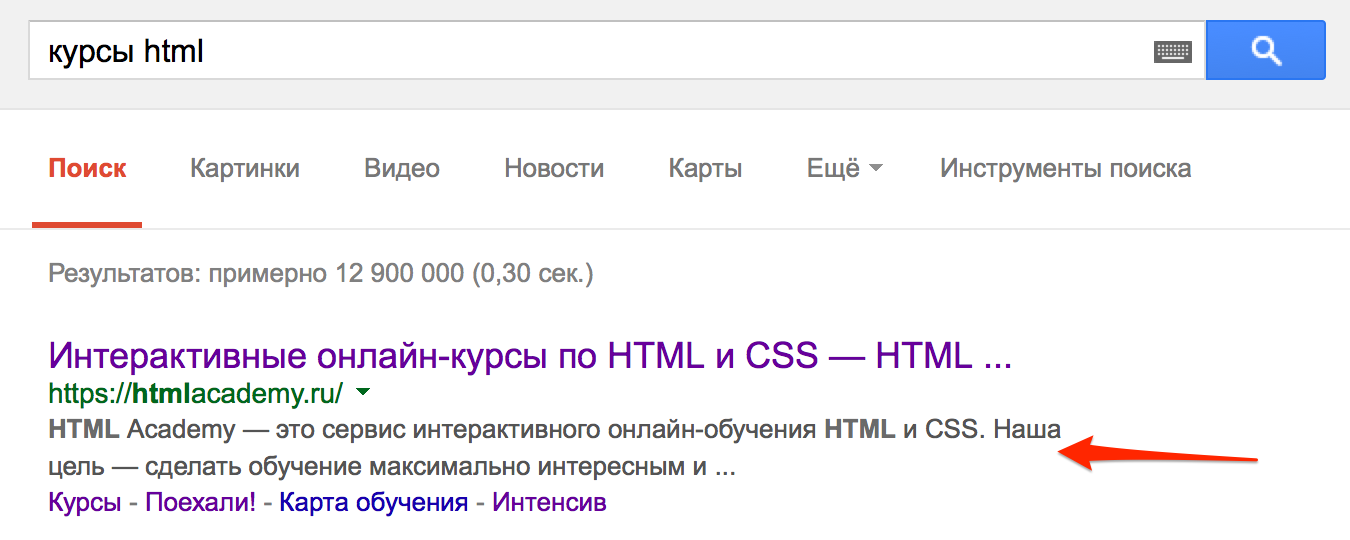
В атрибуте content через запятую перечисляются самые важные слова из содержания страницы. Раньше этот тег был очень важен для поисковиков. Каково положение дел сейчас — большой секрет Яндекса и Гугла.

## Описание содержания страницы

Ещё один полезный для поисковых систем мета-тег — краткое описание страницы. Оно задаётся так:

<meta name="description" content="краткое описание">

В атрибуте content должно быть краткое содержание или аннотация страницы. Оно часто используется поисковиками при отображении результатов поиска.



Пойманный нами инженер из Яндекса не признался, важен ли этот тег для ранжирования, но дал ссылку на [рекомендации по составлению описаний](https://help.yandex.ru/webmaster/recommendations/using-meta-desc.xml). Инженера из Гугла мы ещё только выслеживаем, так что следите за новостями.

## Подключение стилей

Стили можно писать внутри HTML-кода страницы или подключать их как внешний файл.

В первом случае стили называются «встроенными» или «инлайновыми», а писать их нужно внутри тега <style>. Этот тег обычно размещают внутри <head>. Например:

<head>

<style>

CSS-код

</style>

</head>

Внутри <style> пишут обычный CSS-код.

Инлайновые стили используют не так часто, например, для оптимизации скорости загрузки страницы. Чаще используют внешние стили, c которыми мы познакомимся позже.

## Подключение внешних стилей

Чаще всего стили подключают из внешнего файла с расширением .css. Для этого используется тег <link>. Например:

<head>

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

</head>

В атрибуте href задают адрес файла, а атрибут rel="stylesheet" говорит браузеру, что мы подключаем стили, а не что-то другое.

Лучше подключать стили внутри <head>, но это необязательно. Тег <link> будет работать и в другом месте страницы.

В этом задании вы подключите внешний стилевой файл, который расположен по адресу [external.css](https://htmlacademy.ru/assets/courses/26/external.css) (перейдите по ссылке, чтобы открыть этот файл в браузере).

## Подключение скриптов

В вебе следующее разделение ролей: HTML отвечает за структуру документа, стили — за его внешний вид, а скрипты — за поведение. С помощью скриптов, например, можно «оживлять» страницу, добавляя анимацию и другие эффекты. Скрипты создаются с помощью языка JavaScript.

Скрипты подключаются так же, как и стили: их либо пишут внутри страницы, либо подключают как внешние файлы.

Встроенные скрипты пишут внутри тега <script>. Например:

<script>

JavaScript-код

</script>

Тег <script> можно использовать в любом месте HTML-документа, но лучше вставлять его в самом конце перед закрывающим тегом </body>.

Часть возможностей JavaScript постепенно переходит в CSS, например, возможность задавать плавное изменение значений свойств. Вы увидите это в задании.

## Подключение внешних скриптов

Скрипты чаще всего подключают из внешних файлов с расширением .js. Для этого используют тег <script> с атрибутом src, в котором указывается путь к файлу. Например:

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

Обратите внимание, что тег <script> парный. Если вы подключаете внешние скрипты, то просто ничего не пишите внутри тега.

Внешние скрипты лучше подключать перед закрывающим тегом </body>.

В этом задании вы подключите внешний скрипт, который расположен по адресу: [external.js](https://htmlacademy.ru/assets/courses/26/external.js) (перейдите по ссылке, чтобы открыть этот файл в браузере).

# Разметка текста

## Неупорядоченный список

Списки часто используются в различных документах. Иногда, чтобы сделать список, пользователь просто нумерует строчки текста. Такой подход не является хорошим, так как в документе отсутствует логическая сущность «список».

В HTML существует семейство тегов для создания списков: неупорядоченных, упорядоченных и списков определений. В последующих заданиях мы будем тренироваться работать с ними.

Неупорядоченные (или маркированные) списки создаются с помощью тега <ul>, который может содержать внутри себя теги <li>, обозначающие «элемент списка».

## Упорядоченный список

Упорядоченный список создаётся с помощью тега <ol>, который может содержать внутри себя теги <li>.

Если элементы неупорядоченного списка по умолчанию отмечаются маркерами, то элементы упорядоченного списка — нумеруются.

Для упорядоченного списка можно задать атрибут start, который изменяет начало нумерации. Например, код:

<ol start="3">

<li>раз</li>

<li>два</li>

</ol>

Приведёт к такому результату:

1. раз
2. два

## Многоуровневый список

Создать многоуровневый список достаточно просто.

Сначала нужно создать список первого уровня, а затем внутрь любого элемента этого списка, между тегами <li> и </li>, добавить список второго уровня. При этом необходимо аккуратно закрывать все теги.

Пример правильного кода:

<ul>

<li>1

<ul>

<li>1.1</li>

<li>1.2</li>

</ul>

</li>

<li>2</li>

</ul>

Пример кода с ошибкой:

<ul>

<li>1</li>

<ul>

<li>1.1</li>

<li>1.2</li>

</ul>

<li>2</li>

</ul>

## Список определений

Список определений создаётся с помощью трёх тегов:

1. <dl> обозначает сам список определений;
2. <dt> обозначает термин;
3. <dd> обозначает определение термина.

Теги <dt> и <dd> пишутся парами внутри <dl>.

Например:

<dl>

<dt>Термин</dt>

<dd>Определение</dd>

<dt>Второй термин</dt>

<dd>И его определение</dd>

<dt>Кошка</dt>

<dd>Шерстяное изделие развлекательного характера</dd>

</dl>

## Важность. Теги strong и b

Ещё раз отметим, что этот курс посвящён **логической** разметке текста, поэтому уделяется особое внимание смыслу элементов, их предназначению, а не визуальному форматированию.

В предыдущих заданиях вы познакомились с элементами, которые предназначены для разметки крупных блоков текста: заголовков, абзацев и списков. В этом и последующих заданиях мы познакомимся с элементами, предназначенными для разметки небольших фраз и отдельных слов.

Первые два тега предназначены, чтобы указать на важность слова или фразы.

Тег <strong> определяет **важность** отмеченного текста.

Тег <b> предназначен для выделения текста без придания ему особой важности.

Визуально оба тега одинаковы, они выделяют текст полужирным.

Лучше всего отличия этих тегов будут заметны людям, которые используют специальные настройки ОС, в частности, слепым и слабовидящим. Когда они включают функцию чтения текста, то «говорилка» будет интонацией выделять слова с тегом <strong>. То же самое касается и тегов <em> и <i>. Тег <em> «говорилка» будет выделять интонацией.

Отметим, что новый смысл тегу <b> придали в HTML5. Раньше это был тег, который просто делает текст полужирным. То есть он был предназначен только для визуального форматирования.

## Акцентируем внимание. Теги em и i

Следующие два тега предназначены для акцентирования внимания на слово или фразу.

Тег <em> определяет текст, на который сделан особый акцент, меняющий смысл предложения.

Например, если мы хотим подчеркнуть, что Кекс не любит питаться укропом (он больше за тунца), а любит только гонять его по полу, то разметим текст так:

Инструктор Кекс любит <em>играть</em> с укропом.

Тег <i> обозначает текст, который отличается от окружающего текста, но не является более важным. Обычно так выделяют *названия*, *термины*, *иностранные слова*.

Например, если мы хотим указать, что *инспектор* — это какой-то специальный термин, то разметим текст так:

Обычно Кекс пользовался <i>инспектором</i> браузера для поиска ошибок.

Визуально оба тега одинаковы, они выделяют текст курсивом.

Новый смысл тегу <i> придали в HTML5. Раньше это был просто тег для выделения текста курсивом.

## Переносы и разделители. Теги br и hr

Иногда возникает необходимость вставить в текст перенос строки, не создавая при этом абзац. Например, при разметке стихов или текстов песен.

Для этого в HTML предусмотрен одиночный тег <br>.

Иногда этот тег используется для разбиения текста на «как бы абзацы», что является плохим подходом. Используйте для разметки абзацев тег <p>.

Одиночный тег <hr> используется для того, чтобы создать горизонтальную линию-разделитель. На внешний вид этой линии можно влиять с помощью атрибутов, но правильней делать это с помощью CSS (это будет изучаться в последующих курсах).

## Цитаты

В HTML существует несколько тегов для обозначения цитат:

* <blockquote> предназначен для выделения длинных цитат, которые могут состоять из нескольких абзацев. Тег выделяет цитату как отдельный блок текста с отступами.
* <q> предназначен для выделения коротких цитат в предложениях. Текст внутри этого тега автоматически обрамляется кавычками.
* <cite> используется для того, чтобы выделить источник цитаты, название произведения или автора цитаты.

## Верхние и нижние индексы

Следующие два тега обычно используются не для выделения слов, а для выделения отдельных символов. Их используют для указания единиц измерения или для написания простых формул.

Например: 20м2, H2O, X3+X2=1

Тег <sup> отображает текст в виде верхнего индекса.

Тег <sub> отображает текст в виде нижнего индекса.

Стоит отметить, что эти теги являются чисто презентационными и не имеют собственной семантики.

Эти теги можно использовать внутри друг друга для создания более сложных формул.

Если вам нужно вставить очень сложную формулу в HTML-документ, лучше воспользоваться специальным языком разметки [MathML](http://ru.wikipedia.org/wiki/MathML).

## Помечаем изменения. Теги del и ins

Любой документ на протяжении своей «жизни» может изменяться. С распространением динамических веб-приложений вносить изменения в HTML-документы стало проще простого.

Иногда возникает вопрос: а что же именно было изменено в документе, что было добавлено, а что удалено?

Как раз для описания изменений предназначены теги <del> и <ins>.

<del> выделяет текст, который был удалён в новой версии документа.

<ins> выделяет текст, который был добавлен в новой версии документа.

Оба тега имеют атрибут datetime, в котором можно указать дату и время, когда была внесена та или иная правка.

Простейшим примером применения этих тегов может служить список ошибок. Когда ошибка исправлена, её помечают тегом <del>, если найдена новая ошибка, то её добавляют в список и помечают тегом <ins>.

Атрибут datetime предназначен не для людей, а для компьютеров, поэтому дату и время там пишут в стандартизованном формате. При такой разметке программам легче разбирать документы и анализировать, когда произошли те или иные изменения.

## Преформатированный текст

Наверное, вы уже заметили, как отличается отображение кода в HTML-редакторе и в мини-браузере.

Вы можете ставить сколько угодно пробелов в HTML-коде, но браузер отобразит их как один. Вы также можете ставить сколько угодно переносов строки в HTML-коде, а в браузере переноса не будет, если только не использовать специальные теги, например <p>или <br>.

Изменить это поведение браузера можно с помощью тега <pre>, который обозначает «предварительно отформатированный текст». Браузер сохраняет и отображает все пробелы и переносы, которые есть внутри тега <pre>.

Наиболее часто этот тег используется при отображении примеров кода.

## Просто выделенный текст

В HTML5 появился новый тег <mark>, который обозначает выделенный текст.

Иногда при работе с объёмными текстами мы используем маркер, чтобы выделять ключевые слова, идеи или что-то другое на что стоит обратить внимание. Такое же назначение и у тега <mark>.

В современных браузерах текст внутри <mark> подсвечивается жёлтым фоном.

# Ссылки и изображения

## Задаём адрес ссылки

Тег <a> без адреса бесполезен, так как он описывает ссылку, которая никуда не ведёт. Поэтому в предыдущем задании текст никак не изменялся, даже после добавления тегов.

Адрес ссылки задаётся с помощью атрибута href:

<a href="http://keksby.ru">The Great Keksby</a>

У ссылки в примере задан адрес http://keksby.ru.

Значением атрибута является [URL](http://ru.wikipedia.org/wiki/URL), который обычно называют просто адрес. Адреса бывают разные: абсолютные, относительные, указывающие на страницу, на файл, изображение, содержащие якорь и так далее. А значит и ссылка может указывать на любой объект в интернете.

Кстати, вы уже использовали URL, когда [подключали внешние файлы стилей](https://htmlacademy.ru/courses/26/run/9) в курсе «Структура HTML-документа».

## адреса-заглушки

Обычно при вёрстке макетов верстальщик не знает, какие адреса будут у ссылок, поэтому использует так называемые «адреса-заглушки». Они выглядят как атрибут href со значением #, то есть это пустой якорь.

В этой части Кексби-курса вам предстоит добавлять много ссылок. Используйте для них адреса-заглушки, если в задании явно не указан адрес.

Если не задавать ссылкам атрибут href, то они не работают как ссылки и не подчёркиваются. Ссылка без href обозначает заглушку, вместо которой может появиться работающая ссылка. Например, на главной странице можно обернуть логотип не в div, а в ссылку без href:

<!-- Логотип на главной странице -->

<a class="header-logo"><img></a>

<!-- Логотип на других страницах -->

<a class="header-logo" href="index.html"><img></a>

В заголовках статей есть информация об источниках, которая содержит ссылки на другие сайты. Если ссылка ведёт на страницу своего сайта, то её называют внутренней, а если она ведёт на другой сайт, то внешней. Эти внешние ссылки мы сейчас и разметим.

## Ссылки могут открываться в текущей или в новой вкладке браузера.

Ссылки могут открываться в текущей или в новой вкладке браузера. За это поведение отвечает атрибут target тега a и его значения:

* \_self – ссылки открываются в той же вкладке, это значение по умолчанию.
* \_blank – ссылки открываются в новой вкладке (или в новом окне браузера, если он не поддерживает вкладки).

Это не все значения атрибута, ещё есть \_parent и \_top, которые ведут себя почти как \_self и уже давно нигде не используются.

Есть много споров о том, как должны открываться ссылки: в новой вкладке или в текущей. Но мы не будем вдаваться в подробности, так как наш заказчик, Кекс, уже всё решил: «Если ссылка внешняя, то пусть открывается в новом окне, чтобы с моего сайта не уходили, мур-мяу и точка!»

## Абсолютные адреса

Поговорим поподробнее об адресах. Они могут быть абсолютными и относительными.

Абсолютные адреса содержат в себе протокол, имя сервера и путь. Например, в адресе https://htmlacademy.ru/courses:

https:// — это *протокол*

htmlacademy.ru — *имя сервера*, также называется *домен* или *хост*

/courses — *путь*

Абсолютный адрес хорош тем, что однозначно указывает расположение документа. Браузер просто запрашивает по указанному протоколу с указанного сервера документ с указанным путём.

Иногда абсолютные адреса записываются в укороченном виде, например вот так: /courses.

В этом случае, браузер подставляет протокол и сервер текущей страницы. Например, если на сайте https://htmlacademy.ru есть ссылка с адресом /courses, то браузер для запроса преобразует её в такую: https://htmlacademy.ru/courses.

Используйте укороченные абсолютные адреса при задании ссылок внутри своего сайта, так как в случае изменения домена сайта вам не придётся ничего менять.

## Относительные адреса

В относительных адресах нет ни протокола, ни имени сервера, а путь не начинается со слэша /. Вот примеры относительных адресов:

courses/1

./courses

../../run/1

В относительных адресах могут использоваться специальные символы, аналогичные символам в путях файловых систем: . и ...

Если браузер видит, что у ссылки задан относительный адрес, то он должен преобразовать этот адрес в абсолютный, чтобы знать, куда ведёт ссылка. Для этого браузер использует текущий адрес страницы. Например, так преобразуются адреса разных ссылок на одной и той же странице:

|  |  |  |
| --- | --- | --- |
| Текущий адрес | Адрес в ссылке | Преобразуется в |
| http://site.ru/news/1 | 2 | http://site.ru/news/2 |
| http://site.ru/news/1 | .. | http://site.ru/ |
| http://site.ru/users/profile/1 | ../../contacts | http://site.ru/contacts |

Использовать относительные адреса для навигации по сайту не рекомендуется. Однако относительные адреса бывают полезны, например, во внешних CSS-файлах.

Хотите писать JavaScript, используя современный синтаксис (ES2015, ES2016, ES2017)? Уметь тестировать свои программы? Записывайтесь на [продвинутый интенсив по JavaScript](https://htmlacademy.ru/intensive/ecmascript).

## Ссылка с якорем

Ссылки с якорем обычно используются для создания навигации внутри страницы. Например, оглавления в начале страницы с большой статьёй.

Ссылка с якорем содержит символ #, после которого идёт идентификатор. Идентификатор создаётся с помощью атрибута id, который может быть задан у любого тега.

Можно задать адрес, состоящий из одного якоря, например:

<a href="#glava1">Глава 1</a>

При переходе по такой ссылке браузер найдёт на странице элемент с атрибутом id со значением glava1 и прокрутит окно страницы к нему. То есть перезагрузки страницы не произойдёт.

Якорь можно использовать и в абсолютных адресах, тогда после перехода на нужную страницу произойдёт прокрутка к заданной части этой страницы.

## Всплывающая подсказка

Для того, чтобы добавить ссылке всплывающую подсказку, надо использовать атрибут title. Например:

<a title="Подсказка" href="#">

Подсказка появится, когда курсор задержится над ссылкой некоторое время.

Подсказки помогают разъяснить назначение непонятных ссылок, а также ссылок-изображений.

## Добавим изображение

Чтобы добавить на страницу изображение, нужно использовать одиночный тег <img> с атрибутом src, в котором указан адрес картинки. Например:

<img src="logo.png">

В этом задании мы добавим на страницу изображение кота из [шестого задания](https://htmlacademy.ru/courses/40/run/6).

Кстати, самыми распространёнными форматами изображений в сети являются [JPEG](http://ru.wikipedia.org/wiki/JPEG) и [PNG](http://ru.wikipedia.org/wiki/PNG).

## Размеры изображения

Чтобы управлять шириной или высотой изображения, нужно использовать атрибуты widthи height. Пример:

<img width="100" src="...">

В примере изображению задана ширина 100px. Обратите внимание, что в атрибуте widthпосле цифры нет px. Если вы задаёте размер картинки в пикселях, то используйте просто цифры. Добавлять pxне нужно, таков стандарт.

Во втором примере изображению задана относительная ширина, 50 процентов:

<img width="50%" src="...">

Высоту в процентах обычно не задают.

Если задать только один из размеров, ширину или высоту, то вторую размерность браузер вычислит самостоятельно исходя из пропорций изображения.

Если же задать и ширину, и высоту для картинки:

<img width="100" height="100" src="...">

То браузер может нарушить пропорции исходного изображения.

## старайтесь всегда указывать реальные размеры изображений

В [предыдущем задании](https://htmlacademy.ru/courses/104/run/9) вы добавили картинку для первого товара, а теперь добавьте по аналогии изображения остальных товаров.

В каждом задании мы просим вас указывать размеры изображений с помомощью атрибутов width и height тега img. Но зачем это нужно, ведь размер картинок от этого не меняется?

Указывая размеры картинок в атрибутах (и не просто размеры, а настоящие размеры картинок) мы помогаем браузерам быстрее и эффективнее отрисовывать наши страницы. Если браузер знает размеры картинки из HTML-кода, то он может выделить место под неё на странице ещё до загрузки картинки. А когда такая картинка загрузится, браузеру не придётся заново перерисовывать всю страницу.

Так что старайтесь всегда указывать реальные размеры изображений, которые вы вставляете через тег img в соответствующих атрибутах этого тега.

## Изображение-ссылка

Если у пользователя отключены изображения или их невозможно загрузить, то в браузере отображается альтернативный текст. Например, если меню сделано с помощью изображений, то альтернативный текст поможет понять, куда ведёт каждый пункт. В общем, задавать альтернативный текст хорошо.

Альтернативный текст изображения задаётся с помощью атрибута alt. Пример:

<img src="cat.png" alt="Кот в полном расцвете сил">

## Изображение-ссылка

Ссылки можно делать не только с помощью текста, но и с помощью изображений. Для этого нужно обернуть тег <img> в тег <a>. Например:

<a href="http://keksby.ru">

<img src="cat.png" alt="Кекс">

</a>

Часто ссылки-изображения используются в галереях, когда с уменьшенной версии изображения ставится ссылка на полноразмерное изображение.

# Знакомство с таблицами

## Простейшая таблица

Из всех объектов, которые используются для разметки текста в интернете, таблицы являются самыми сложными для новичков. Действительно, табличные данные приходится публиковать достаточно часто. В отличие от списков, абзацев, заголовков, изображений с таблицами всегда возникает море проблем.

В этом курсе мы узнаем, как с помощью HTML описывать таблицы, научимся делать простые и достаточно сложные таблицы. И, самое главное, научимся аккуратно оформлять таблицы с помощью CSS. Вы увидите, как на самом деле легко и просто работать с таблицами.

Начнём с самого простого. Уберите комментарий в коде редактора и посмотрите на простейшую таблицу из четырёх строк и двух колонок.

## Добавляем строки

Простейшая таблица описывается с помощью трёх тегов:

1. <table> обозначает таблицу.
2. <tr> расшифровывается как *«table row»*, обозначает строку таблицы.
3. <td> расшифровывается как *«table data»*, обозначает ячейку внутри строки таблицы.

Теги <td> располагаются внутри тегов <tr>, а те, в свою очередь, внутри <table>. Почти всё текстовое содержимое таблицы размещается внутри тегов <td>.

В простейшей таблице в каждой строке должно быть одинаковое количество ячеек, то есть внутри всех <tr> должно быть одинаковое количество <td>.

Потренируйтесь добавлять строки в таблицу.

## Добавляем столбцы

Со строками справились, теперь потренируемся добавлять в таблицу столбцы.

Для того, чтобы добавить столбец в таблицу, надо в каждую строку <tr> добавить по ячейке <td>.

В этом задании вам нужно будет в исходную таблицу добавить ещё два столбца. То есть в каждой строке должно быть по четыре ячейки.

## Задаём рамки с помощью CSS

Вы научились создавать простые таблицы, добавлять в них любое количество строк и столбцов. Теперь пришло время оформить эти таблицы.

Таблицы в предыдущих заданиях отображались с рамками по умолчанию. Такие рамки отображаются, если у тега <table> задан атрибут border с ненулевым значением.

Но с помощью атрибута border гибко управлять рамками не получается. С его помощью можно только изменять их толщину.

Поэтому мы будем учиться использовать CSS. С помощью CSS-свойства border можно задавать как внешние рамки таблицы, так и рамки каждой ячейки.

Потренируемся использовать CSS для задания рамок таблицы.

## Улучшаем отображение рамок

Мы задали рамки таблицы с помощью CSS, но они не так хороши, как хотелось бы. По умолчанию браузер рисует рамки таблицы и рамки отдельных ячеек раздельно, это отлично видно на примере.

Чтобы избавиться от таких двойных рамок, используется CSS-свойство таблицы border-collapse. Вот так:

table {

border-collapse: collapse;

}

Значение collapse убирает двойные рамки: cхлопываются рамки соседних ячеек, а также рамки ячеек и внешняя рамка таблицы. При этом внешняя рамка таблицы может исчезнуть, и чтобы её вернуть, можно увеличить её ширину.

## Горизонтальные и вертикальные рамки

Иногда требуется, чтобы рамки ячеек в таблице отображались не полностью. Например, чтобы отображалась только нижняя рамка ячеек, тогда таблица получается расчерченной по горизонтали. Аналогично, если отображать только боковые рамки ячеек, то таблица получается разбитой на столбцы.

Такие эффекты легко достигаются с помощью CSS. Для этого необходимо использовать не свойство border, которое задаёт рамки для всех сторон ячейки, а одно из свойств:

* border-top,
* border-right,
* border-bottom,
* border-left.

Эти свойства задают отображение только одной рамки ячейки: верхней, правой, нижней или левой соответственно.

В этом задании вы потренируетесь создавать горизонтально и вертикально расчерченные таблицы.

## Отступы внутри ячеек

Вы освоили простейшие приёмы для работы с рамками таблиц. Наша таблица уже смотрится аккуратно, но содержимое ячеек прилипает к рамкам. Если добавить отступы внутри ячеек, то информация будет восприниматься намного лучше.

Отступы внутри ячеек можно добавлять с помощью атрибута cellpadding тега <table>. Но лучше его не использовать, а задавать отступы с помощью CSS.

CSS-свойство padding задаёт «внутренние отступы элемента» со всех сторон. Можно задавать отступы для каждой из сторон отдельно, используя свойства:

* padding-top,
* padding-right,
* padding-bottom,
* padding-left.

Например, чтобы задать у ячеек все отступы в 10 пикселей, а отступ слева в 20 пикселей, нужно написать такой CSS-код:

td {

padding: 10px;

padding-left: 20px;

}

## Отступы между ячейками

Большинство задач по оформлению таблиц решаются с помощью работы с рамками, отступами внутри ячеек, изменения цвета фона ячеек.

Помимо внутренних отступов можно задавать отступы между ячейками таблицы.

Отступы между ячейками не работают с border-collapse: collapse, что достаточно логично, ведь рамки ячеек в этом режиме «склеены» и их не разорвать.

Поэтому в этом задании мы используем border-collapse: separate, которое «расклеивает» ячейки. На самом деле это значение по умолчанию, а мы используем его только для наглядности. Если удалить свойство border-collapse, то результат не изменится, ячейки будут отображаться раздельно.

Отступы между ячейками можно задать:

* с помощью атрибута cellspacing тега <table>
* или c помощью CSS-свойства border-spacing.

Отметим, что свойство border-spacing задаётся для таблицы, в отличие от padding, которое задаётся для ячеек.

## Ячейки-заголовки

Поздравляем! Вы научились создавать простейшие таблицы и оформлять их. В следующей части курса мы познакомимся с ещё несколькими табличными тегами и научимся создавать более сложные таблицы.

Обычно в таблицах выделяют названия столбцов или строк. В HTML для этого предусмотрен специальный тег <th>, который расшифровывается как *«table header»*и обозначает ячейку-заголовок. Тег <th> аналогичен <td>, он так же должен располагаться внутри <tr>, для него стилями можно задавать все те же свойства.

По умолчанию текст внутри <th> выделяется жирным и выравнивается по центру ячейки.

В этом задании мы наконец-то добавим в нашу таблицу ячейки-заголовки и оформим их: добавим увеличенные отступы и более яркие рамки.

## Заголовок таблицы

В предыдущем задании вы создали заголовочные ячейки с помощью подходящих тегов, в этом задании мы создадим подпись к таблице (или заголовок таблицы).

Сейчас подпись Распределение браузеров сделана с помощью заголовка первого уровня. В принципе, можно догадаться, что это название таблицы, но [семантически](https://ru.wikipedia.org/wiki/%D0%A1%D0%B5%D0%BC%D0%B0%D0%BD%D1%82%D0%B8%D1%87%D0%B5%D1%81%D0%BA%D0%B0%D1%8F_%D0%B2%D1%91%D1%80%D1%81%D1%82%D0%BA%D0%B0) правильнее будет сделать это название с помощью тега <caption>.

Тег <caption> должен размещаться внутри тега <table>, причём непосредственно внутри него и первым, до остальных вложенных тегов. Вот так:

<table>

<caption>Текст</caption>

...

</table>

Тег заголовка идёт первым внутри таблицы, но с помощью CSS можно переместить заголовок таблицы в любое место: сверху или снизу таблицы, по центру, справа или слева.

По вертикали заголовок таблицы перемещается CSS-свойством caption-sideсо значениями top и bottom, которые обозначают до и после таблицы соответственно.

По горизонтали заголовок таблицы выравнивается CSS-свойством text-alignсо значениями left, right и center.

## Объединяем ячейки в строках

Мы подобрались к одному из самых сложных вопросов по работе с таблицами. Это объединение ячеек.

Когда вы объединяете ячейки в текстовом редакторе, например, Word, то программа многое делает за вас. В чистом HTML задача объединения сложнее, однако, не стоит пугаться, сам принцип объединения ячеек не так уж сложен, просто нужно больше внимательности.

Исходная таблица в примере не очень красива, но с помощью объединения ячеек мы сделаем сложный заголовок таблицы, вот такой:



Начнём с объединения ячеек по горизонтали. Чтобы объединить ячейки по горизонтали, необходимо использовать атрибут colspan у тегов <th> или <td>.

Когда вы задаёте ячейке атрибут colspan со значением 2, то ячейка как бы «растягивается» на ячейку справа, но та ячейка не исчезает, а отодвигается и в таблице появляется новый столбец. Чтобы удалить его, нужно удалить ячейку, которая находится справа от «растянутой».

## Объединяем ячейки в столбцах

Объединение ячеек по вертикали немного сложнее. Оно осуществляется с помощью атрибута rowspan у тега <td> или <th>.

Когда вы задаёте ячейке атрибут rowspan со значением 2, то ячейка как бы «растягивается» на следующую строку. При этом ячейка, которая была под «растянутой» отодвигается в своей же строке вправо, что добавляет в таблицу лишний столбец. Удалив ячейку, которая была под «растянутой» мы избавимся от этого столбца.

В задании всё будет очень наглядно. Мы объединим ячейку «Браузер» с ячейкой снизу, а затем поправим CSS-код, чтобы получить тот результат, который изображён на картинке:



## Выравнивание содержимого в ячейках

Содержимое ячеек можно выравнивать по горизонтали и по вертикали с помощью CSS.

За выравнивание по горизонтали отвечает CSS-свойство text-align. Чаще всего используются значения left, center и right.

За выравнивание по вертикали отвечает CSS-свойство vertical-align. Чаще всего используются значения top, middle и bottom.

На самом деле, значений у обоих свойств больше, но в случае с ячейками нас интересуют только перечисленные.

Чтобы задать выравнивание содержимого ячеек, надо в стилях указать:

{

vertical-align: значение;

text-align: значение;

}

Эти стили повлияют на все ячейки. Чтобы задать выравнивание только в определённых ячейках, нужно назначить им классы и определить стили для классов. В этом примере для вас уже подготовлены ячейки с классами.

## Добавим цвета

В этом задании мы раскрасим нашу скучную чёрно-белую таблицу. Таблицы можно раскрашивать, задавая цвет фона ячеек, цвет текста в ячейках, а также цвет рамок. Можно задавать и фоновые изображения, но это мы разберём в последующих курсах.

Цветами можно управлять с помощью этих свойств:

* background-color — задаёт цвет фона,
* color — цвет текста,
* border-color — цвет рамок.

До этого мы использовали компактную форму для описания рамок: border: 1px solid lightgray. В этой записи цвет задаёт третья часть — lightgray.

Значения цветов в CSS задаются [разными способами](https://developer.mozilla.org/en/docs/Web/CSS/color_value). Мы будем использовать *ключевые слова* для описания цвета.

Таким образом, чтобы задать цвета для ячейки в CSS, нужен такой код:

td {

color: цвет;

background-color: цвет;

border: 1px solid цвет;

}

Конечно, раскрашивать можно и td, и th, и даже table. Приступим!

## Раскрашиваем строки

В предыдущем задании вы раскрасили ячейки и ячейки-заголовки с помощью CSS. Если задавать стили для тега, например, th или td, то они применятся ко всем тегам. В нашем примере все ячейки окрасились в светло-жёлтый, а все ячейки-заголовки в светло-голубой.

Как быть, если стили нужно задать для какой-то определённой ячейки, группы ячеек, или строки? Можно использовать классы и применять стили для этих классов. Например, вот так:

.my-class {

стили

}

В этом задании мы раскрасим в разные цвета строки таблицы. Для этого воспользуемся классами, которые уже заданы для строк в примере.

## Задаём размеры таблицы

По умолчанию ширина и высота таблицы зависит от содержимого и отступов внутри ячеек. Чем меньше содержимого, тем меньше размеры таблицы.

С помощью CSS можно управлять размерами таблицы, задавать желаемую ширину и высоту. Также размерами можно управлять с помощью атрибутов таблицы, но мы рассмотрим только CSS.

Стоит отметить, что у таблицы есть минимальные размеры, которые зависят от содержания, меньше которых она не сожмётся, какое бы значение ширины или высоты ни задавалось.

Ширина таблицы задаётся с помощью CSS-свойства width, а высота с помощью свойства height, например:

table {

width: 100px;

height: 100px;

}

Размеры таблицы можно задавать как в абсолютных единицах, например, в пикселях — 20px, так и в относительных, в процентах — 20%.

При использовании процентов размеры таблицы будут вычисляться с учётом размеров родительского элемента, в нашем случае окна мини-браузера.

Особое значение auto включает расчёт размеров по умолчанию. Например, width: auto;или height: auto;.

Важное замечание. Проценты при задании высоты обычно не работают.

## Задаём размеры отдельных ячеек и столбцов

Размеры ячеек и столбцов тоже можно задавать вручную, особенно если вам не нравится, как браузер распределил ширину колонок. Размеры ячеек задаются точно так же, как и размеры таблицы: с помощью CSS-свойств width и height.

Есть два варианта добавления стилей ячейкам:

1. Назначать ячейкам уникальные имена классов, например, class="cell-11", и применять стили для этих классов.
2. Использовать атрибут style, внутри которого можно писать CSS-код.

Пример второго варианта:

<td style="width: 100px;">

...

</td>

К счастью, редко нужно задавать размеры каждой ячейки. Обычно размеры ячеек прописывают, когда надо вручную установить ширину столбцов таблицы: для этого достаточно задать ширину для каждой ячейки из первой строки.

В этом задании вы зададите ширину столбцов, используя атрибут style. А также увидите отличия между шириной столбцов по умолчанию и заданной вручную.

# Знакомство с формами

## Первая форма

Формы нужны для того, чтобы отправлять данные с веб-страницы на веб-сервер, который сможет эти данные обработать: зарегистрировать пользователя, создать сообщение на форуме, отправить письмо и так далее. В общем, формы в вебе просто необходимы.

Чтобы создать форму, нужно использовать парный тег <form>, внутри которого размещаются поля формы. У тега <form> есть два важных атрибута:

* action задаёт адрес, URL, отправки формы;
* method задаёт метод отправки формы.

Пример:

<form action="https://echo.htmlacademy.ru" method="get">

поля формы

</form>

Для отправки формы обычно используют методы get или post. Если не указать атрибут method, то будет использован get.

Метод get посылает данные формы в строке запроса, то есть они видны в адресной строке браузера и следуют после знака вопроса. Например:

https://www.google.com/search?q=htmlacademy

Метод get лучше использовать в поисковых формах, потому что он позволяет получить ссылку на результаты поиска и передать её кому-то.

Метод post посылает данные в теле HTTP-запроса и используется, когда нужно отправить много данных и ссылка на результат обработки этих данных не нужна. Например, при редактировании личного профиля.

## Текстовое поле ввода

Большинство полей форм создаётся с помощью одиночного тега <input>. У этого тега два обязательных атрибута:

* type задаёт тип поля;
* name задаёт имя поля.

Тип поля влияет на то, как оно будет отображаться и вести себя. Самый распространённый тип — это text, который обозначает текстовое поле. Он же используется по умолчанию. Пример:

<form action="https://echo.htmlacademy.ru" method="get">

<input type="text" name="search">

</form>

Имя поля нужно, чтобы правильно обработать данные на сервере. Обычно, имя поля должно быть уникальным в пределах формы, хотя есть исключения. Для задания имени поля используют латинские буквы и цифры.

## Идентификатор и значение по умолчанию

Атрибут id поля ввода обозначает идентификатор. Он должен быть уникальным не только в пределах формы, но и на всей странице.

Обычно идентификаторы используют для повышения удобства работы с формой, например, создают подписи, связанные с мелкими полями. При нажатии на такие подписи активируется связанное поле. И это удобно, так как по большой подписи попасть легче, чем по маленькому полю. Также идентификаторы используют в JavaScript для работы с полями.

Идентификатор в отличие от имени поля не передаётся на сервер. Лучше использовать идентификаторы, отличающиеся от имени поля, особенно актуально это для полей множественного выбора, которые мы разберём далее в курсе.

Атрибут value задаёт значение поля ввода по умолчанию. Это тоже повышает удобство.

Согласитесь, приятно зайти в огромную анкету на каких-нибудь госуслугах, а там ваши паспортные данные уже подставлены в нужные поля и заполнять их не надо. И всё благодаря тому, что программист добавил к полям атрибут value с нужными данными.

## Подпись для поля ввода

Можно подумать, что сделать подпись к полю очень просто. Пишем текст рядом с полем и всё готово:

Подпись <input type="text" name="username">

На самом деле этого недостаточно — мы получили просто кусок текста и поле, которые расположены рядом друг с другом, но логически никак не связаны.

Есть специальный тег, который позволяет смело сказать: «Этот кусок текста действительно подпись к этому полю!». Это парный тег <label>.

Он связывает текст и поле ввода логически. А ещё если нажать на текст в такой подписи, то курсор переместится в соответствующее поле.

Создавать подписи к полям с помощью <label> — хороший приём. Используйте его.

Первый способ создать подпись — просто обернуть текст подписи и тег поля в тег <label>, вот так:

<label>

Подпись <input type="text" name="username">

</label>

Надо отметить, что при оборачивании текста в тег <label> он визуально никак не меняется, ведь главная задача подписи — создать логическую связь.

## Связываем подпись и поле по id

Иногда обернуть поле и текст подписи в тег <label> нельзя. Например, когда они размещены в разных ячейках таблицы.

В этом случае можно связать подпись с полем с помощью атрибута id. Алгоритм такой:

1. Добавляем к полю ввода идентификатор с помощью атрибута id.
2. Оборачиваем текст подписи в тег <label>.
3. Добавляем тегу <label> атрибут for.
4. В атрибут for записываем такое же значение, что и в атрибуте id у поля.

Например:

<label for="user-field-id">Имя пользователя</label>

...

*много-много других тегов*

...

<input id="user-field-id" type="text" name="username">

## Добавим ещё одно поле

Пришло время добавить в форму ещё одно поле.

Сделайте это по всем правилам хорошего тона: помимо имени задайте полю идентификатор, добавьте подпись к полю и свяжите их.

Мы могли бы связать поле и подпись, просто обернув их в тег <label>. Но в этом задании мы тренируем более сложный вариант — с использованием id и for.

## Поле для ввода пароля

Мы создаём простую форму входа. Уже есть два поля и связанные с ними подписи. Одно из полей предназначено для ввода пароля, но сейчас оно является простым текстовым полем.

Чтобы сделать его настоящим полем для ввода пароля, в котором текст будет отображаться «звёздочками», нужно просто изменить значение атрибута type на password.

## Кнопка отправки формы

Форма практически готова. Осталось добавить кнопку для отправки формы. Такая кнопка создаётся с помощью тега <input> c типом submit.

Надпись на кнопке можно задать с помощью атрибута value. Для кнопки отправки формы задавать имя необязательно. Но если имя задано, то на сервер будут отправляться имя и значение кнопки.

Обычно имя для кнопки отправки задают, когда в форме несколько кнопок, отвечающих за разные действия. Браузер отправляет на сервер имя и значение только той из них, на которую нажал пользователь. Таким образом, сервер может понять, какую кнопку нажали и что нужно сделать.

## Многострочное поле ввода

Мы научились создавать простейшие формы с текстовыми полями и кнопками. А теперь познакомимся с более сложными элементами формы.

Многострочное текстовое поле создаётся с помощью парного тега <textarea>. У него есть атрибуты name и id, которые аналогичны атрибутам текстового поля.

Атрибут rows принимает целочисленное значение и задаёт высоту многострочного поля в строках. Атрибут cols задаёт ширину поля в символах. В качестве ширины символа берётся некоторая «усреднённая ширина».

Атрибут value у многострочного поля отсутствует, а значение по умолчанию задаётся по-другому. Текст, расположенный между открывающим и закрывающим тегом <textarea>и является значением по умолчанию. Вот так:

<textarea>Значение по умолчанию</textarea>

## Чекбокс или «галочка»

Поле-галочка — это тег <input> с типом checkbox.

Галочка работает по принципу «либо да, либо нет». Если галочка стоит, то браузер посылает переменную с именем поля на сервер, если галочки нет, то не посылается ничего. Таким образом, атрибут value не является обязательным.

Чтобы галочка стояла по умолчанию, нужно добавить к тегу атрибут checked. Вот так:

<input type="checkbox" checked>

Чекбокс не подразумевает выбор одного элемента из нескольких. Поэтому если в одной форме есть несколько чекбоксов, то имена у них должны быть разными.

## Переключатель или «радиобаттон»

Поле-переключатель — это тег <input> с типом radio.

Обычно переключатели размещают группами по несколько штук. Причём у переключателей из одной группы должно быть одинаковое имя и разные значения, которые задаются c помощью value.

Таким образом, атрибут value является для переключателей обязательным. Браузер отправляет на сервер значение value выбранного переключателя.

В этом задании мы начнём создавать переключатель.

Перейти к заданию

* **index.html**
* style.css

## группа переключателей

Теперь добавим ещё один вариант ответа в наш переключатель. Для этого нужно добавить ещё один <input> с таким же именем, но другим значением value.

Подобным образом можно создавать группы переключателей с любым количеством вариантов.

Чтобы сделать какой-либо вариант в переключателе выбранным по умолчанию, нужно добавить к соответствующему тегу <input> атрибут checked, как у поля-галочки.

Кстати, имя поля у переключателей одной группы должно быть одинаковым, но идентификаторы всегда должны быть уникальными.

## Раскрывающийся список или «селект»

Раскрывающийся список так же, как и переключатель, позволяет выбрать один вариант ответа из нескольких.

Раскрывающийся список создаётся с помощью парного тега <select>, у которого есть знакомые атрибуты name и id.

Варианты ответов задаются с помощью парных тегов <option>, которые должны располагаться внутри тега <select>. Например:

<select name="theme">

<option value="light">Светлая тема</option>

<option value="dark">Тёмная тема</option>

...

</select>

В атрибуте value тега <option> задаётся значение варианта ответа, а внутри этого тега располагается подпись варианта ответа.

Если при отправке формы у выбранного варианта задан value, то на сервер отправится значение этого атрибута. В противном случае будет отправлен текст подписи.

## «Мультиселект»

Раскрывающийся список можно превратить в так называемый «мультиселект», то есть список, в котором можно выбрать не один, а несколько вариантов.

Чтобы сделать это, нужно добавить к тегу <select> атрибут multiple.

Выбрать несколько вариантов можно, щёлкая по ним с зажатой клавишей Ctrl на Windows или Command на MacOS.

Высоту мультиселекта можно изменять с помощью атрибута size тега <select>.

Чтобы отметить как выбранные по умолчанию одно или несколько значений, нужно к соответствующим тегам <option> добавить атрибут selected.

При отправке данных мультиселекта на сервер с PHP после имени в значении атрибута name ставятся символы квадратных скобок []. Например, <select name="days[]">. Это необязательное требование для имени мультиселекта, а нужно только для корректной обработки данных в PHP.

## Поле для загрузки файлов

Поле для загрузки файлов — это тег <input> с типом file. Для этого поля обязательным атрибутом является имя.

Чтобы поле заработало и браузер смог передать выбранный файл на сервер, необходимо добавить форме атрибут enctype со значением multipart/form-data. Не полю, а форме.

Кстати, внешний вид таких полей очень сильно отличается в зависимости от операционной системы и очень плохо изменяется с помощью стилей.

## Скрытое поле

И ещё одно невидимое и очень полезное поле. Это скрытое поле. Его используют, когда в форме нужно отправить какие-то дополнительные служебные данные, которые не вводятся пользователем.

Например, это могут быть реквизиты заказа или номер пользователя в форме оплаты.

Скрытое поле — это тег <input> с типом hidden.

# [Знакомство с CSS](https://htmlacademy.ru/courses/41)

## CSS-правила

Весь CSS-код состоит из повторяющихся блоков следующего вида:

селектор {

свойство1: значение;

свойство2: значение;

}

Такой блок называется «CSS-правило». Каждое CSS-правило содержит хотя бы один селектор и свойство.

Простейшие селекторы — это селекторы по именам тегов. С их помощью можно задать стили для всех абзацев на странице, для всех ссылок, заголовков первого уровня и так далее. Такие селекторы содержат имя тега без символов < и >. Например:

p {

/\* стили для абзацев \*/

}

h1 {

/\* стили для заголовков \*/

}

## Продвинутые селекторы

К более сложным селекторам можно отнести селекторы с использованием классов и псевдоклассов.

Класс позволяет объединять разные элементы в смысловые группы и применять к ним одинаковое оформление. Например, можно создать класс «элементы с ошибкой» и задать ему красный цвет текста. Затем можно добавлять этот класс к любому HTML-тегу: абзацу, заголовку, элементу списка и так далее.

Класс тега можно задать с помощью атрибута class, который содержит имя класса (или имена классов через пробел). Пример:

<p class="help"></p>

<p class="help error"></p>

В примере у первого абзаца задан класс help, у второго абзаца заданы классы helpи error.

Селектор с использованием класса задаётся так: .имя\_класса. Например:

.help { … }

.error { … }

## Каскадность и приоритеты

Когда в предыдущем задании мы задали цвет фона для правила с классом truth, одно из свойств второго абзаца переопределилось:

p {

padding: 10px;

background-color: #dff0d8;

}

+

.truth {

background-color: #aaddff;

}

=

стили второго абзаца {

padding: 10px;

background-color: #aaddff;

}

Когда для одного и того же элемента есть несколько CSS-правил с одинаковыми свойствами, браузер использует понятия приоритетов и специфичности, чтобы выбрать значение свойства из нескольких возможных. Упрощённо, можно сказать что:

1. CSS-правила в значении атрибута style самые приоритетные,
2. за ними идёт селектор с id,
3. затем селектор с классом,
4. затем селектор с именем тега.

На самом деле, механизм определения приоритетов и специфичности более сложный. Подробно он описан в курсе «[Наследование и каскадирование](https://htmlacademy.ru/courses/66)».

## Наследование

Другой важный механизм CSS — это наследование. Он заключается в том, что часть стилей может передаваться от родительского элемента к дочерним (вложенным в него).

Например, все элементы внутри тега body являются дочерними по отношению к нему. Если для body в стилях задать цвет текста красным, то цвет всех остальных элементов тоже станет красным.

Ещё пример: тег ul является родительским по отношению к вложенным в него тегам li. Если задать для ul шрифт курсивом, то и внутри всех li шрифт станет курсивным.

## Ненаследуемые свойства

Наследование работает не для всех свойств. Некоторые свойства применяются только к самому элементу и не переходят к его потомкам.

К таким ненаследуемым свойствам относятся: ширина, высота, отступы, режим позиционирования и другие. Согласитесь, было бы странно задать отступы для bodyи обнаружить, что у всех вложенных элементов тоже появились отступы.

Более подробно наследование описано в курсе «[Наследование и каскадирование](https://htmlacademy.ru/courses/66)».

# Селекторы, часть 1

## Селекторы по классам

Класс — это один из атрибутов тегов. Выглядит он вот так:

<li class="first"></li>

Этот атрибут особенный, так как в CSS существует возможность выбирать элементы по классу. Делается это с помощью такого селектора: .*имя\_класса*. Например:

.first {

/\* стили для класса *first* \*/

}

Имена классов могут состоять из латинских символов, цифр и знаков - и \_. Имя класса должно начинаться с латинской буквы.

## Контекстные селекторы

Селектор может состоять из нескольких частей, разделённых пробелом, например:

p strong { ... }

ul .hit { ... }

.footer .menu a { ... }

Такие селекторы называют *контекстными* или *вложенными*. Их используют для того, чтобы применить стили к элементу, только если он вложен в нужный элемент.

Например, селектор .menu a сработает для ссылки a только в том случае, если она расположена внутри элемента с классом .menu.

Читать их проще всего справа налево:

/\* выбрать все теги strong внутри тегов p \*/

p strong { ... }

/\* выбрать все элементы с классом .hit внутри тегов ul \*/

ul .hit { ... }

/\* выбрать все ссылки внутри элементов с классом .menu,

которые лежат внутри элементов с классом .footer \*/

.footer .menu a { ... }

Таким образом, можно задавать элементам различные стили в зависимости от их контекста. Если ссылка расположена внутри меню, сделать её крупнее, а если внутри основного текста, то задать ей нужный цвет.

В этом задании вы потренируетесь использовать контекстные селекторы.

Х

## Соседние селекторы

Контекстные селекторы используются для вложенных друг в друга элементов, а соседние — для расположенных рядом.

Например, теги <li> в списке являются соседними по отношению друг к другу и вложенными в тег <ul>.

Соседние селекторы записываются с помощью знака +, например, селектор1 + селектор2. Стили применятся к элементу, подходящему под селектор2, только если сразу перед ним расположен элемент, подходящий под селектор1.

Пример. Есть два элемента списка:

<li class="hit"></li>

<li class="miss"></li>

Селектор .hit + .miss применит стили к элементу с классом miss, так как перед ним есть элемент с классом hit.

Селектор .hit + li тоже применит стили к элементу с классом miss, а селектор .miss + .hitне сработает.

## Контекстные и соседние селекторы

Селекторы в CSS можно очень гибко комбинировать. В частности, можно комбинировать контекстные и соседние селекторы.

Например, селектор .player-1 .hit + .miss сработает для тега с классом miss, если сразу перед ним расположен тег с классом hit и оба тега расположены внутри тега с классом player-1.

В этом задании вы потренируетесь комбинировать контекстные и соседние селекторы. Класс биатлониста будет задавать контекст, а соседние селекторы будут использоваться для выбора мишени.

## Дочерние селекторы

Потомком называются любые элементы, расположенные внутри родительского элемента. А дочерними элементами называются ближайшие потомки. Взгляните на пример:

<ul>

<li><span>...</span></li>

<li><span>...</span></li>

</ul>

По отношению к <ul><li> являются дочерними элементами и потомками, а <span> — потомки, но не дочерние элементы.

Контекстные селекторы влияют на всех потомков, что не всегда удобно. Иногда необходимо задать стили только для дочерних элементов. Особенно это полезно при работе с [многоуровневыми списками](https://htmlacademy.ru/courses/38/run/5).

Для этого существует дочерний селектор, в котором используется символ >. Например: ul > li или ul > li > span.

В этом задании вы разберётесь, чем отличаются контекстные и дочерние селекторы.

## Псевдоклассы

Псевдоклассы — это дополнения к обычным селекторам, которые делают их ещё точнее и мощнее. Обычный селектор — это снайперский прицел, а с псевдоклассом он становится прибором ночного видения.

Псевдокласс добавляется к селектору c помощью символа :, вот так селектор:псевдокласс. Например:

a:visited { ... }

li:last-child { ... }

.alert:hover { ... }

Знакомство с псевдоклассами мы начнём с first-child и last-child.

Псевдокласс first-child позволяет выбрать первый дочерний элемент родителя, а last-child — последний дочерний элемент. Например:

li:last-child { ... }

Этот селектор выберет последний элемент списка.

## Псевдокласс :nth-child

Псевдоклассы из предыдущего примера относятся к семейству псевдоклассов, помогающих выбирать элементы по их расположению.

Вспомним [задание 4](https://htmlacademy.ru/courses/42/run/4). В нём каждому тегу <li> был задан собственный класс. Используя классы, мы могли выбрать любой из пяти тегов. Если бы тегов было десять, то пришлось бы использовать десять разных классов.

С помощью псевдокласса nth-child можно выбирать теги по порядковому номеру, не используя классы. Синтаксис псевдокласса: селектор:nth-child(выражение). Выражениемможет быть число или формула. Например:

1. li:nth-child(2) { ... }

2. li:nth-child(4) { ... }

3. li:nth-child(2n) { ... }

Первый селектор выберет второй элемент списка, второй селектор — четвёртый элемент списка, третий селектор — все чётные элементы.

Подробнее об :nth-child и синтаксисе его выражений рассказано в демо-курсе [Использование псевдокласса :nth-child](https://htmlacademy.ru/demos/2).

## :nth-child и контекстные селекторы

Селекторы с псевдоклассами хорошо сочетаются с контекстными селекторами.

Например, селектор:

.shooter-2 li:nth-child(3) { ... }

Выберет третий тег <li> внутри блока с классом shooter-2.

В этом задании вам нужно будет сделать то же, что и в [задании 5](https://htmlacademy.ru/courses/42/run/5), но без использования классов для мишеней.

## Псевдокласс :hover

Некоторые псевдоклассы позволяют выбирать элементы, с которыми взаимодействует пользователь. Сначала познакомимся с псевдоклассом :hover.

Этот псевдокласс позволяет выбрать элемент, когда на него наведён курсор мыши и кнопка мыши не нажата. Примеры:

1. a:hover { ... }

2. tr:hover { ... }

3 .menu-item:hover { ... }

Первый селектор выбирает ссылку, второй строку таблицы, третий элемент с классом menu-item, но только в том случае, если на них наведён курсор мыши.

Благодаря этому псевдоклассу можно добавлять в интерфейс динамику и интерактивность, так как элементы начинают реагировать на действия пользователя, изменяя свой внешний вид.

## Динамические эффекты с помощью :hover

Интересовались ли вы, как с помощью CSS создаются выпадающие меню?

Львиная доля динамических эффектов, создаваемых с помощью CSS, опираются на несколько псевдоклассов, главный из которых, конечно же, :hover. Весь секрет заключается в сочетании контекстных селекторов и псевдоклассов. Посмотрите на пример:

li.top ul.submenu {

display: none;

}

li.top:hover ul.submenu {

display: block;

}

Первое правило прячет список-подменю. Второе правило гласит: «если на верхний пункт меню, в котором находится подменю, наведут курсор, то надо показать подменю». Вот так всё просто.

Общий принцип такой: родительский элемент реагирует на наведение мыши и изменяет свойства элементов-потомков. То есть всё работает на контекстных селекторах вида селектор1:hover селектор2.

## Псевдоклассы :link, :visited и :active

Отвлечёмся от биатлона и познакомимся с псевдоклассами для ссылок.

* :link выбирает ещё не посещённые ссылки.
* :visited выбирает посещённые ссылки.
* :active выбирает активные ссылки (кнопка мыши зажата на ссылке).

Пример задания CSS-правил для ссылок:

a:link { ... }

a:visited { ... }

a:hover { ... }

a:active { ... }

Обратите внимание на порядок правил. Если их расположить по-другому, то некоторые могут не сработать.

## Псевдокласс :focus

Псевдокласс :focus позволяет выбрать элемент, который в данный момент в фокусе. Например, текстовое поле, в которое установлен курсор, находится в фокусе.

В фокусе могут быть не только текстовые поля. Если вы переключаетесь между элементами веб-страницы с помощью клавиши **tab**, то в фокус будут попадать ссылки.

Пример использования псевдокласса:

input:focus {

/\* стили для поля в фокусе \*/

}

## Селекторы атрибутов

Ранее мы познакомились с атрибутом class и специальными селекторами по классу. Существуют селекторы, которые позволяют выбирать элементы по любым атрибутам.

Чаще всего такие селекторы используются [при работе с формами](https://htmlacademy.ru/courses/50), так как поля форм имеют атрибут type с разными значениями.

Селекторы атрибутов записываются с использованием квадратных скобок: элемент[атрибут]. Примеры селекторов:

1. input[checked] { ... }

2. input[type="text"] { ... }

Первый селектор выберет поля формы, у которых есть атрибут checked, второй селектор выберет поля формы, у которых атрибут type имеет значение text.

## Селектор по id

Существует ещё один HTML-атрибут, для которого существует специальный селектор. Этот атрибут id (идентификатор), а селектор записывается с помощью символа #, например, #some-id.

На значение id распространяются те же ограничения, что и на имя класса. Также id должен быть уникальным на странице.

Использование селекторов по id при оформлении считается плохой практикой. Существуют редкие исключения из этого правила, например, при оживлении слайдера на чистом CSS.

# Наследование и каскадирование

## Иерархическое дерево

HTML-документ представляет собой иерархическое дерево. Это означает, что у каждого элемента (кроме корневого) есть только один родитель, т.е. элемент, внутри которого он располагается. У корневого раздела родитель отсутствует. Рассмотрим простейшую страницу:

<html>

<head></head>

<body>

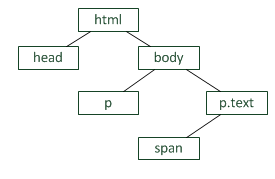
<p>Текст документа</p>

<p class="text">Выделенная <span>строка</span></p>

</body>

</html>

Для этой страницы можно нарисовать такое иерархическое дерево:



Оно схематически отображает структуру вложенности элементов. В данном примере видно, что у элемента span родителем является p.text, а у p.text родитель — body.

Иерархическая структура документа определяет основы концепции наследования.

## Наследование

Наследование в CSS — механизм, с помощью которого значения свойств элемента-родителя передаются его элементам-потомкам.

Стили, присвоенные некоторому элементу, наследуются всеми потомками (вложенными элементами), если они не переопределены явно. Например, размер шрифта и его цвет достаточно применить к body, чтобы все элементы внутри имели те же свойства.

Наследование позволяет сократить размер таблицы стилей, но если стилей много, то отследить какой родительский элемент установил некоторое свойство, становится сложнее.

## Наследование «на пальцах»

Давайте на простых примерах подробнее разберёмся, в чём же преимущество наследования.

Рассмотрим пример:

<p class="text">Cтрока c выделенным <span>словом</span></p>

Представим, что нам нужно установить красный цвет текста для всего текста. Зададим CSS-свойства следующим образом:

.text {

color: red;

}

Благодаря наследованию цвет текста в теге span автоматически станет красным:

Cтрока c выделенным словом

А так бы выглядел результат, если бы наследование не работало:

Cтрока c выделенным словом

Нам пришлось бы отдельно прописывать цвет текста для тега span. И тогда установка таких простых свойств как стиль шрифта стала бы большой проблемой: нужно было бы задавать свойства для всех возможных вложенных тегов.

## Ещё немного про наследование

Наверняка вы обращали внимание, что не все свойства наследуются тегами-потомками от их родителей.

Действительно, было бы странно, если бы свойство border автоматически устанавливалось для всех вложенных элементов.

Например, для этого куска кода:

<p class="bordered">Cтрока c выделенным <span>словом</span></p>

Установим CSS-свойство:

.bordered {

border: 1px solid green;

}

Если бы наследовались все свойства, то результат бы выглядел так:

Cтрока c выделенным словом

На самом деле граница будет нарисована только у тега p.

О том, какие именно свойства наследуются, мы расскажем в следующих заданиях.

## Наследуемые свойства

К наследуемым свойствам относятся в первую очередь свойства, определяющие параметры отображения текста:

font-size, font-family, font-style, font-weight, color, text-align, text-transform, text-indent, line-height, letter-spacing, word-spacing, white-space, direction и т. д.

Также к наследуемым свойствам относятся list-style, cursor, visibility, border-collapseи некоторые другие. Но они используются значительно реже.

Весь список наследуемых свойств смотрите в [стандарте CSS](https://www.w3.org/TR/CSS21/propidx.html). Значение yes в колонке Inherited?.

Эти свойства можно и нужно задавать через предков, следуя семантике документа.

Например, параметры текста зачастую не меняются в пределах отдельных блоков страницы: меню, основного содержания, информационных панелей. Поэтому общие параметры текста (цвет, размер, гарнитура) обычно указывают в стилях самих блоков.

## Ненаследуемые свойства

В предыдущем задании мы перечислили основные наследуемые свойства. Все остальные относятся к ненаследуемым. Это параметры позиционирования, размеров, отступов, фона, рамок и т. д.

А именно: background, border, padding, margin, width, height, position и др.

Весь список ненаследуемых свойств смотрите в [стандарте CSS](https://www.w3.org/TR/CSS21/propidx.html). Значение no в колонке Inherited?.

Не наследуются они из соображений здравого смысла. Например: если для какого-либо блока установлен внутренний отступ, автоматически выставлять такой же отступ каждому вложенному элементу нет никакой надобности. Эти параметры чаще всего уникальны для каждого отдельного блока.

## Принудительное наследование

Для каждого свойства может быть задано значение inherit.

Оно означает, что данное свойство принимает такое же значение, как и у родительского элемента. Значение inherit может быть использовано для усиления наследуемых значений, а также в свойствах, которые обычно не наследуются.

Запись выглядит следующим образом:

p {

background: inherit;

}

В данном случае у тегов p свойство background будет таким же, как и у их родительских тегов.

## Каскадирование

CSS расшифровывается как «*Cascading* Style Sheets» или «*каскадные* таблицы стилей».

Каскадность обозначает, что к одному и тому же элементу может применяться несколько CSS-правил (наборов CSS-свойств). Среди этих свойств могут быть и конфликтующие между собой. Поэтому существуют инструкции, которые определяют, каким будет финальный набор свойств элемента.

Например, для элемента:

<p class="text" style="color: red;"></p>

CSS-правила существуют как минимум в трёх разных местах:

1. в подключаемом файле style.css для селекторов p или .text;
2. в атрибуте style;
3. в стандартных стилях отображения, встроенных в браузер.

Каскадирование как раз и определяет, какие именно свойства из этих источников применятся к данному абзацу.

Имеется три основные концепции, управляющие порядком, в котором применяются CSS-свойства:

1. важность;
2. специфичность;
3. порядок исходного кода.

Самыми важными для нас являются последние две и подробнее о них мы поговорим отдельно.

## Битва за курочку

Перед тем как разбирать сложные правила работы каскадирования, расчёта специфичности и определения приоритетов, потренируемся на котиках.

Разберём пример. Вот HTML-код, в котором есть абзац с двумя классами:

<p class="red blue">Синий или красный?</p>

А вот CSS-код c двумя правилами для этих классов:

.blue {

color: blue;

}

.red {

color: red;

}

Вопрос: какого цвета будет текст абзаца? Какое из CSS-правил приоритетнее?

Ответ: красного цвета, второе правило приоритетнее.

Это происходит потому, что селекторы у правил одинакового типа, и соответственно обладают одинаковой специфичностью. В таком случае более высоким приоритетом обладает то правило, которое расположено в CSS-коде ниже. В нашем случае это правило для класса red.

В этой серии заданий вам нельзя менять значения CSS-свойств, но можно изменять селекторы или менять порядок CSS-правил в коде.

## Битва за курочку. Раунд второй

Чуть более сложный пример. Тот же HTML:

<p class="red blue">Синий или красный?</p>

Немного другой CSS:

p.blue {

color: blue;

}

.red {

color: red;

}

В этом случае текст абзаца будет синим. Происходит это потому, что селектор p.blueболее специфичный, чем селектор .red.

Простое объяснение специфичности звучит так:

Чем меньшее количество элементов потенциально может выбрать селектор, тем он специфичнее.

В нашем примере селектор .red выберет все теги с нужным классом, а селектор p.blueвыберет только абзацы с нужным классом.

Заметьте, что в этом задании одно из CSS-правил вынесено в заблокированный HTML-код, а вам нужно победить его, усилив другое CSS-правило.

## Битва за курочку. Раунд третий

А теперь посмотрим, как ведут себя контекстные селекторы.

<div class="experiment">

<p class="red blue">Синий или красный?</p>

</div>

И в CSS используем контекстный селектор для второго правила:

p.blue {

color: blue;

}

.experiment .red {

color: red;

}

Какой теперь будет цвет текста?

Кексик и Рудольф помогут разобраться, какой селектор специфичнее.

## Битва за курочку. Борьба накаляется

Как вы уже знаете, существуют селекторы не только по классам, но и по id. Они начинаются с решётки #.

HTML:

<div id="experiment-1" class="experiment">

<p class="red blue">Синий или красный?</p>

</div>

CSS:

#experiment-1 .blue {

color: blue;

}

.experiment .red {

color: red;

}

Особенность атрибута id заключается в том, что его значение должно быть уникальным в пределах страницы. То есть, может существовать только один тег с определенным значением id.

Получается, что селектор по id может выбрать только один элемент. И поэтому он на порядок специфичнее селекторов по тегам, классам, а также комбинаций этих селекторов.

Кексик и Рудольф демонстрируют его работу.

## Битва за курочку. Запрещённый приём

Битва подходит к концу, и чтобы удержать победу, Рудольф решается на запрещённый приём: он прописывает стили с помощью атрибута style прямо в теге курочки.

CSS-правила, которые прописаны в style обладают наивысшим приоритетом. Такой способ задания стилей не приветствуется в профессиональной вёрстке сайтов и годится только для создания быстрых прототипов. Поэтому мы и назвали этот приём запрещённым.

Однако существует способ переопределить из подключаемых CSS-файлов даже стили, заданные в атрибуте style. Для этого нужно использовать ключевое слово !important. Оно задаёт CSS-свойству усиленный приоритет. Вот пример:

HTML:

<p style="color: red;" class="blue">Синий или красный?</p>

CSS:

.blue {

color: blue !important;

}

Цвет текста в этом примере будет синим.

При вёрстке не рекомендуется часто использовать !important. По возможности старайтесь обходиться без него.

## Ещё одна задачка на специфичность

Как вы уже догадались, самым главным механизмом для определения приоритетов стилей является специфичность. Поэтому давайте еще немного потренируемся работать с ней.

Рассмотрим пример: на полу в коробке сидит кот

<div id="floor">

<span class="cat-in-box">Кексик</span>

</div>

Допустим в стилях существуют следующие определения:

span {

background-color: #27ae60; /\* Зелёный \*/

}

div span {

background-color: #2980b9; /\* Синий \*/

}

#floor .cat-in-box {

background-color: #34495e; /\* Мокрый асфальт \*/

}

.cat-in-box {

background-color: #8e44ad; /\* Фиолетовый \*/

}

#floor span {

background-color: #c0392b; /\* Красный \*/

}

div .cat-in-box {

background-color: #e67e22; /\* Оранжевый \*/

}

А теперь вопрос на засыпку: какого цвета будет коробка? Сначала сделайте предположение, а затем проверьте.

Почему именно такой и как это определяется мы расскажем в следующем задании.

## Расчёт значения специфичности

Вы вдоволь наигрались со специфичностью, а теперь пришло время изучить полные правила её вычисления.

Специфичность селектора разбивается на 4 группы — a, b, c, d:

* если стиль встроенный, т.е. определен как style="...", то а=1, иначе a=0;
* значение b равно количеству идентификаторов (тех, которые начинаются с #) в селекторе;
* значение c равно количеству классов, псевдоклассов и селекторов атрибутов;
* значение d равно количеству селекторов типов элементов и псевдо-элементов.

После этого полученное значение приводится к числу (обычно в десятичной системе счисления). Селектор, обладающий большим значением специфичности, обладает и большим приоритетом.

Посчитаем специфичность в нашем примере:

|  |  |  |
| --- | --- | --- |
| Селектор | a, b, c, d | Число |
| span | 0, 0, 0, 1 | 1 |
| div .cat-in-box | 0, 0, 1, 1 | 11 |
| **#floor .cat-in-box** | **0, 1, 1, 0** | **110** |
| div span | 0, 0, 0, 2 | 2 |
| .cat-in-box | 0, 0, 1, 0 | 10 |
| #floor span | 0, 1, 0, 1 | 101 |

Отсюда сразу видно, что в нашем примере самым приоритетным является селектор #floor .cat-in-box.

## Перекрестное наследование

При создании стилей для сходных по внешнему виду или функциональности элементов, которые могут использоваться на странице неоднократно, очень удобно пользоваться перекрёстным наследованием.

Приём этот заключается в следующем:

1. создается базовый стиль для таких элементов;
2. определяются вспомогательные стили, которые применяются к элементам по мере надобности;
3. элемент наследует базовый стиль и один или несколько вспомогательных.

**Пример:**

На странице используются кнопки разного назначения: для отправки форм, для сброса информации в полях формы, как элементы навигации и т.д.

Можно вынести общее оформление (размеры, отступы и т.д.) для всех кнопок в отдельное CSS-правило для класса, например, .button.

А затем создать дополнительные CSS-правила, в которых будут определены только различающиеся свойства этих кнопок, например, цвет фона. Для этих правил можно использовать такие названия классов: .button-send, .button-clear, .button-navigation.

Каждая кнопка в HTML-коде будет иметь два класса: общий и дополнительный.

<a class="button button-send">Отправить</a>

# Оформление текста, часть 1

## Главный текстовый тег — span

При оформлении текста с помощью css чаще всего используют тег <span>. Он обозначает «просто текстовый блок». То есть особенного собственного смысла он не имеет. Также этот тег никак не изменяет отображение текста.

Однако, дополнительный смысл данному тегу добавляют с помощью классов. Например:

<span class="error"></span>

<span class="ok"></span>

А уже для класса с помощью css задают стили и тем самым изменяют оформление.

## Свойство font-size: задаём размер шрифта

Cвойство font-size задает размер шрифта.

Размер шрифта лучше всего задавать в em — относительной единице измерения. 1emобычно равен длине буквы M в данном шрифте.

Другие единицы измерения для задания размеров шрифта:

* Пиксели: 20px
* Пункты: 15pt
* Проценты: 80%

Пиксели и пункты — это абсолютные единицы измерения, а проценты — относительные.

Также размер шрифта можно задавать с помощью ключевых слов: small, large и т.д. Но их обычно не используют.

## Свойство font-weight: толщина начертания

Полужирность текста можно задавать с помощью свойства font-weight, которое имеет два основных значения:

1. normal — обычное начертание;
2. bold — полужирное начертание.

На самом деле это свойство имеет много значений: bold, bolder, lighter, normal, 100, 200, 300, 400, 500, 600, 700, 800, 900. Эти значения задают степень толщины шрифта, от самого тонкого, до самого толстого.

Но большинство браузеров всё равно умеют отображать только два варианта толщины: обычный и полужирный. Поэтому и остальные значения свойства обычно не используют.

## Свойство font-style: курсив

Начертание текста можно задавать с помощью свойства font-style. Его основные значения:

1. normal — обычное начертание;
2. italic — курсивное начертание.

Это свойство имеет и другие значения, но их почти не используют.

## Свойство font-family: шрифт

Задать семейство шрифта можно с помощью свойства font-family. Можно задавать конкретное название шрифта: "Times New Roman". А можно задавать желаемый тип шрифта, например:

* serif — шрифт с засечками;
* sans-serif — шрифт без засечек.

Есть и другие типы, но они используются реже.

Обычно в качестве значения свойства задают список шрифтов, перечисляя их через запятую. В начале списка располагают самый редкий шрифт, затем похожий, но более распространенный, а в самом конце списка — желаемый тип шрифта. Пример:

body {

font-family: "PT Sans", "Arial", serif;

}

Браузер проходит по списку слева направо и использует первый найденный в системе шрифт.

## Свойство color: цвет текста

Цвет текста задаётся с помощью свойства color. Значение цвета можно задавать разными способами:

* Шестнадцатеричным кодом, например #ff9900.
* Ключевым словом: red.
* В RGB-формате: rgb(255, 255, 0).

Чаще всего цвет задают в шестнадцатеричном формате.

## Свойство text-decoration: подчёркивание и другие эффекты

Дополнительное оформление текста можно задать с помощью свойства text-decoration. Его значения:

1. underline — подчёркивание;
2. line-through — зачёркивание;
3. overline — черта сверху;
4. none — убирает вышеперечисленные эффекты.

К тексту можно одновременно применить несколько эффектов, если перечислить значения через пробел.

## Декоративное подчёркивание

Вы, наверное, уже много раз видели красивое пунктирное подчёркивание. Его используют для оформления ссылок и других динамических элементов. Технология следующая:

1. Убираем обычное подчёркивание с помощью text-decoration.
2. Задаём нужный цвет текста с помощью color.
3. Добавляем декоративное подчёркивание с помощью свойства border-bottom.

Также можно при наведении курсора скрывать такое подчёркивание с помощью псевдокласса :hover.

## Задаём регистр символов с помощью text-transform

С помощью css можно управлять даже регистром символов: делать буквы строчными или прописными. Делается это с помощью свойства text-transform. Его значения:

1. lowercase — все строчные;
2. uppercase — все прописные;
3. capitalize — каждое слово начинается с прописной;
4. none — отменяет изменение регистра.

## Управляем пробелами: white-space

Как вы [уже знаете](https://htmlacademy.ru/courses/38/run/14), браузер игнорирует множественные пробелы и переносы строк в HTML-коде. Изменить это поведение можно с помощью тега <pre>.

Однако, с помощью CSS управлять пробелами и переносами можно более гибко. За это отвечает свойство white-space, значения которого:

* nowrap — отображает весь текст одной строкой без переносов;
* pre — сохраняет пробелы и переносы как в исходном коде аналогично тегу <pre>;
* pre-wrap — работает как значение pre, но добавляет автоматические переносы, если текст не помещается в контейнер;
* normal — режим по умолчанию.

## Горизонтальное выравнивание текста: text-align

Выравнивание текста по горизонтали задается с помощью свойства text-align. Его вы уже [использовали](https://htmlacademy.ru/courses/39/run/18) в курсе про таблицы. Значения свойства:

1. left — выравнивание по левому краю;
2. right — по правому краю;
3. center — по центру;
4. justify — по ширине.

## Вертикальное выравнивание: vertical-align

Выравниванием текста по вертикали можно управлять с помощью свойства vertical-align. Его действие хорошо заметно в ячейках таблицы. Внутри текстовой строки «работа» этого свойства заметна, если в ней есть фрагменты разного размера.

У данного свойства много значений, но самые часто используемые:

1. top — выравнивание по верхнему краю строки;
2. middle — по середине;
3. bottom — по нижнему краю;
4. baseline — по базовой линии (значение по умолчанию).

## Верхние и нижние индексы на CSS

С помощью CSS можно имитировать теги <sub> и <sup>, которые применяются для [создания нижних и верхних индексов](https://htmlacademy.ru/courses/38/run/12). Делается это так:

1. Используем свойство vertical-align со значением sub или super.
2. Немного уменьшаем размер шрифта с помощью font-size.

## Свойство line-height: управляем высотой строки

Высотой строки или, правильнее, межстрочным интервалом можно управлять с помощью свойства line-height. Значение этого свойства можно задавать следующими способами:

1. Множителем, например 1.5, 2.
2. В процентах: 150%.
3. С помощью любых других единиц измерения CSS: 12px, 2em.
4. Ключевым словом normal, которое задает автоматический расчет высоты строки.

Предпочтительнее задавать межстрочный интервал либо множителем, либо в относительных единицах измерения.

## Вертикальный ритм текста

В веб-дизайне существует понятие «вертикальный ритм текста». Оно достаточно сложное и мы не будем глубоко в него вдаваться. Отметим лишь, что хороший вертикальный ритм улучшает восприятие текста.

Чтобы сохранить вертикальный ритм и хорошую читабельность текста при любом размере шрифта, нужно задавать размеры шрифта, межстрочный интервал и вертикальные отступы между заголовками и абзацами в относительных единицах.

Тогда, как бы пользователь ни изменял размер шрифта, ваш текст останется читабельным и удобным для восприятия. В этом задании вы потренируетесь использовать относительные единицы измерения.

Дополнительные материалы для тех, кто заинтересовался вертикальным ритмом:[О вёрстке вертикального ритма](https://peredelka.wordpress.com/2013/10/29/%D0%BE-%D0%B2%D0%B5%D1%80%D1%81%D1%82%D0%BA%D0%B5-%D0%B2%D0%B5%D1%80%D1%82%D0%B8%D0%BA%D0%B0%D0%BB%D1%8C%D0%BD%D0%BE%D0%B3%D0%BE-%D1%80%D0%B8%D1%82%D0%BC%D0%B0/),   
[Compose to a Vertical Rhythm](https://24ways.org/2006/compose-to-a-vertical-rhythm/),   
[Создание вертикального ритма на сайте](http://pixelgene.ru/articles/vertical-rhythm.html),   
[Генератор вертикального ритма](https://www.gridlover.net/try).

# [Фоны, часть 1](https://htmlacademy.ru/courses/53)

## Cвойство background-color

Цвет фона можно задать с помощью CSS-свойства background-color.

Напомним, что цвета в CSS можно задавать в разных форматах: в шестнадцатеричном (или HEX), в RGB или RGBA, а также с помощью цветовых констант, таких как red или green.

Более подробно о цветах можно прочитать на странице [HTML Colors](https://www.w3schools.com/html/html_colors.asp) или в переводной статье [о теории цвета на Хабре](https://habr.com/post/189766/).

Вот пример использования свойства:

selector {

background-color: #ff0000;

}

## Свойство background-image

Фоновое изображение можно задать с помощью CSS-свойства background-image. Делается это так:

selector {

background-image: url("адрес-картинки");

}

Адрес картинки обязательно заключается внутрь url("..."). Адреса фоновых картинок такие же, как и адреса обычных картинок, которые вы задавали в курсе про [ссылки и изображения](https://htmlacademy.ru/courses/40).

Элементу можно одновременно задавать и цвет фона, и фоновую картинку. В этом случае картинка будет отображаться поверх фонового цвета.

В этом курсе мы в качестве подопытных картинок будем использовать отличные иконки [Дениса Сажина](http://iconka.com/ru/).

## Свойство background-repeat

По умолчанию фоновое изображение повторяется. Это хорошо заметно, когда оно меньше блока. Управлять этим поведением можно с помощью CSS-свойства background-repeat. У свойства 4 значения:

* repeat — повторять во всех направлениях. Это значение по умолчанию.
* repeat-x — повторять только по горизонтали.
* repeat-y — повторять только по вертикали.
* no-repeat — не повторять.

## Свойство background-position

Cвойство background-position управляет расположением фонового изображения. Значение свойства состоит из двух частей, разделенных пробелом: x y.

x задаёт расположение по горизонтали, а y по вертикали.

В качестве значения x можно использовать ключевые слова left, center, right, значения в процентах и в пикселях.

В качестве значения y можно использовать ключевые слова top, center, bottom, значения в процентах и в пикселях.

Примеры использования свойства:

background-position: 50% 50%;

background-position: right bottom;

background-position: 50px 100px;

background-position: 0 100%;

background-position: left bottom;

Так сработают примеры:

1. картинка будет по центру;
2. правый нижний угол;
3. отступ 50px от левого края и 100px от верхнего;
4. левый нижний угол;
5. левый нижний угол.

## Ещё немного background-position

Когда фоновое изображение больше блока, то оно обрезается. Управлять тем, какую часть изображения будет видно, также можно с помощью свойства background-position.

Иногда для этого удобно использовать относительные значения (проценты), а иногда абсолютные (пиксели).

Кстати, можно использовать не только положительные, но и отрицательные значения. А также комбинировать пиксели и проценты.

## Свойство background-attachment

Обычно фоновое изображение прокручивается вместе с содержимым блока. Это хорошо видно в мини-браузере. Прокрутите окно вниз и увидите, что фон поднимется наверх.

С помощью свойства background-attachment можно зафиксировать фон на месте и он не будет перемещаться при прокручивании.

Значение свойства:

* scroll — фон прокручивается вместе с содержимым. Это значение по умолчанию.
* fixed — фон не прокручивается, зафиксирован на одном месте.

Это тот случай, когда легче показать, чем объяснять: выполняйте задание и смотрите, как изменяется поведение фона при прокрутке.

## Свойство background

Фон элемента можно задавать с помощью отдельных свойств: background-color, background-image и так далее. Получается довольно громоздкая запись.

Также задать фон можно с помощью сокращенного свойства background, в котором через пробел перечисляются его компоненты:

background: [bc] [bi] [br] [bp] [ba];

/\* Обозначения:

[bc] — background-color

[bi] — background-image

[br] — background-repeat

[bp] — background-position

[ba] — background-attachment

\*/

Если какой-то компонент не указан, то берется значение по умолчанию. Ниже примеры.

background: #e74c3c;

background: url("img.png") no-repeat;

background: url("img.png") 10px 20px;

В первом примере просто задан цвет фона.

Во втором примере задано не повторяющееся фоновое изображение, а также по умолчанию прозрачный цвет фона, расположение в левом верхнем углу.

В третьем примере задано фоновое изображение и его расположение, а также по умолчанию прозрачный цвет фона и режим повторения во все стороны.

## Формат JPEG

В последующих нескольких заданиях мы рассмотрим наиболее распространенные форматы изображений для веб. И разберемся, где и как их лучше использовать при вёрстке.

Первый формат — [JPEG](https://ru.wikipedia.org/wiki/JPEG). Этот формат хорошо подходит для фотографий и картин и плохо подходит для схем, чертежей, текста и графики.

При сохранении изображения в формат JPEG можно задавать уровень качества. Это позволяет добиваться снижения веса файла при достаточно хорошем качестве изображения.

Однако, если установить уровень качества слишком низким, то появятся артефакты.

JPEG не поддерживает прозрачность, поэтому изображение в формате JPEG — это всегда прямоугольник. Можно имитировать прозрачность, если в графическом редакторе задать изображению такой же цвет фона, как и у контейнера фотографии.

Используйте JPEG для размещения фото с наилучшим соотношением "размер файла"/"качество".

## Формат PNG-8

На самом деле, формат [PNG](https://ru.wikipedia.org/wiki/PNG) один. А PNG-8 и PNG-24 это его подтипы, а также режимы сохранения в Photoshop. Мы не будем глубоко вдаваться в детали, а разберем типовые особенности.

PNG-8 по характеристикам схож с форматом GIF. Он хорошо подходит для схем, чертежей, графиков и текста, а также изображений, где мало цветов. Максимальное количество цветов — 256.

PNG-8, как и GIF, поддерживает прозрачность. Это означает, что пиксель либо полностью прозрачный, либо полностью непрозрачный.

Если нужно сделать изображение с прозрачностью, то лучше задавать обводку такую же, как цвет фона. В фотошопе при сохранении обводка называется «Mate». Без обводки картинка будет выглядеть «обкусанной». Если фон неоднородный (градиент, 3 блок в мини-браузере), то PNG-8 с обводкой будет смотреться плохо.

Чем PNG лучше GIF? PNG более современный и свободный формат, а GIF более старый и проприетарный (за его использование могут потребовать деньги). Уровень сжатия, качество и другие характеристики примерно одинаковые, поэтому лучше всегда использовать PNG.

## Формат PNG-24

PNG-24 — отличнейший формат. Он хорошо подходит и для схем-чертежей-графиков-текста, и для сложных многоцветных изображений, так как поддерживает практически неограниченное количество цветов.

На фотографиях обычно он уступает JPEG по размеру файла, но превосходит по качеству изображения. Но иногда разница по размеру файла настолько некритична, что можно использовать PNG.

Самый главный плюс PNG-24 — это полноценная поддержка полупрозрачности, которой нет ни в одном другом формате. Поэтому в веб-разработке при вёрстке сложных фонов, графических элементов со сложными тенями и так далее альтернативы PNG-24 просто нет.

Давным-давно в IE была проблема с поддержкой полупрозрачных PNG, но в более поздних версиях IE (8+), таких проблем нет.

Сравните котика-PNG-8 с обводкой из предыдущего задания и котика-PNG-24 из этого задания. Особенно то, как выглядят их края на желтом блоке и блоке с градиентом.

Некоторые верстальщики перестают использовать PNG-8 и используют только PNG-24.

## Формат GIF

Наверное, единственная причина использовать GIF заключается в том, что он поддерживает анимированные изображения.

Во всех остальных случаях используйте PNG или JPEG (если речь идёт о фотографиях и PNG даёт слишком тяжёлый файл).

## Несколько фонов

Несколько фонов одному элементу можно задать с помощью множественных фонов. Эта и другие возможности рассмотрены в продолжении курса — [«Рамки и фоны, часть 2»](https://htmlacademy.ru/courses/88).

А сейчас рассмотрим старую надёжную технику создания нескольких фонов.

Суть техники заключается в том, что мы вкладываем элементы друг в друга и делаем их одинакового размера, а затем каждому элементу задаём свой фон. То есть каждый элемент служит одним слоем фона.

Фоны вложенных элементов перекрывают друг друга: чем глубже элемент, тем выше его фон.

Кстати, для удобства ширину лучше задавать внешнему элементу (так как все вложенные будут той же ширины), а высоту самому глубокому, так как он растянет по высоте всех своих родителей.

## Эффекты с повторяющимся фоном

Повторяющийся фон, тот у которого repeat, repeat-x или repeat-y, часто используется для создания интересных декоративных эффектов.

Например:

* стежки;
* зазубренные края;
* градиенты и тени.

Да, сейчас уже градиенты и тени можно делать с помощью CSS. Но иногда, когда тень или градиент достаточно сложные, проще использовать полупрозрачный PNG.

Самое главное при создании таких эффектов — выбрать картинку с нужным периодом. Она может быть очень маленькой и даст существенную экономию веса страницы.

Попрактикуемся. Только, пожалуйста, изменяйте режим повторения после того, как пропишете фоновые изображения. Так будет нагляднее.

## Спрайты

Спрайт — это одно большое изображение, в котором содержится много маленьких, как бы карта изображений. Вот живой [пример](https://htmlacademy.ru/assets/courses/53/bootstrap-sprites.png) (эта же картинка выведена на тёмном фоне внизу мини-браузера), который мы будем использовать в задании.

Спрайты используются, чтобы снизить количество запросов на сервер. Каждая маленькая картинка — это отдельный запрос, а чем меньше запросов, тем лучше. Поэтому маленькие картинки «склеивают» в одну большую.

Части спрайта отображают в элементах с небольшими размерами. Такому элементу задают картинку-спрайт в качестве фона и смещают её таким образом, чтобы была видна нужная её часть.

В спрайты обычно объединяют иконки и различные мелкие декоративные изображения. Кстати, есть сервисы для быстрого создания спрайтов и генерации CSS-кода для них, например, [SpritePad](https://spritepad.wearekiss.com/).

## Множественные фоновые изображения

 — это очень просто. Берёте знакомое свойство background и перечисляете в нем картинки через запятую. Например:

background:

url("img1.png") no-repeat 0 0,

url("img2.png") repeat-x 50% 50%,

url("img3.png");

В этом примере у элемента будет 3 фоновых картинки. Причем картинка img3.png будет в нижнем слое, а img1.png в верхнем. В множественных фоновых изображениях нельзя задавать фоновый цвет, поэтому он вынесен в отдельный блок.

# Блочная модель документа

## Блочные элементы [1/23]

Элементы HTML страницы обычно делятся на блочные и строчные.

Блочные элементы можно представлять как прямоугольные области на странице. Они имеют следующие особенности:

1. До и после блочного элемента существует перенос строки.
2. Блочным элементам можно задавать ширину, высоту, внутренние и внешние отступы.
3. Занимают всё доступное пространство по горизонтали.

К блочным элементам относятся такие теги как: <p>, <h1>, <h2>, <ul> и так далее.

Еще один важный блочный тег — это тег <div>, который обозначает просто «блок» или «прямоугольный контейнер». Этот тег чаще всего используется для создания сеток.

Существуют и более сложные типы элементов, им будет посвящена вторая часть курса.

## Строчные элементы [2/23]

Строчные элементы располагаются друг за другом в одной строке, при необходимости строка переносится. Особенности строчных элементов:

1. До и после строчного элемента отсутствуют переносы строки.
2. Ширина и высота строчного элемента зависит только от его содержания, задать размеры с помощью CSS нельзя.
3. Можно задавать только горизонтальные отступы.

К строчным элементам относятся такие теги как: <a>, <strong>, <em>, <span> и так далее.

Строчные элементы предназначены для оформления текста на уровне небольших фраз и отдельных слов. Блочные же элементы предназначены для разметки крупных блоков текста (заголовки, абзацы, списки) и создания сетки.

## Ширина и высота [3/23]

Ширина и высота элементов задаются с помощью свойств width и height соответственно.

По умолчанию блочные элементы занимают всю доступную ширину, которая равна ширине родительского контейнера или окна браузера.

Высота по умолчанию блочных элементов зависит от их содержимого. Если задать блочному элементу ширину и высоту так, что содержимое элемента не будет в него помещаться, то оно как бы «выпадет» из него.

Строчные элементы не реагируют на задание ширины и высоты в CSS.

Задать ширину блоку можно, например, так:

.selector {

width: 100px;

height: 100px;

}

Вернуть значения по умолчанию можно с помощью специального значения auto:

.selector {

width: auto;

height: auto;

}

## Внутренние отступы, свойство padding [4/23]

Свойство padding задает внутренние отступы элемента — отступы от внешней границы элемента до его содержания. Эти отступы еще иногда называют полями.

Существует несколько способов записи свойства padding.

padding: 10px;

padding: 5px 10px;

padding: 5px 10px 15px;

padding: 5px 10px 15px 20px;

1. Одинаковые отступы со всех сторон.
2. Отступы сверху и снизу 5px, справа и слева 10px.
3. Отступ сверху 5px, слева и справа 10px, снизу 15px.
4. Разные отступы со всех сторон, в порядке верхний, правый, нижний, левый.

Также можно задавать отступы для разных сторон с помощью свойств padding-left, padding-right, padding-top, padding-bottom.

Для строчных элементов лучше не задавать вертикальных отступов, т.к. они ведут себя непредсказуемо.

В задании внутренние отступы блоков сделаны прозрачными.

## Внешние отступы, свойство margin [5/23]

Свойство margin задает внешние отступы элемента — отступы от внешней границы элемента до границ родительского элемента или до соседних элементов.

Способы записи свойства margin аналогичны свойству padding.

margin: 10px;

margin: 5px 10px;

margin: 5px 10px 15px;

margin: 5px 10px 15px 20px;

1. Одинаковые отступы со всех сторон.
2. Сверху и снизу 5px, справа и слева 10px.
3. Сверху 5px, слева и справа 10px, снизу 15px.
4. Верхний, правый, нижний, левый отступы соответственно.

Можно задавать отступы для разных сторон с помощью свойств margin-left, margin-right, margin-top, margin-bottom.

Строчные элементы реагируют только на горизонтальные отступы.

## Рамки [6/23]

Рамка задаётся с помощью свойства border, которое состоит из трёх компонентов:

1. ширина рамки;
2. стиль рамки;
3. цвет.

Например:

selector{

border: 5px solid red;

}

Это правило задаёт красную сплошную рамку толщиной 5px.

Задавать рамку можно одним свойством border, а можно и с помощью отдельных свойств border-width, border-style, border-color. Например:

selector{

border-width: 5px;

border-style: solid;

border-color: red;

}

Это правило задаёт такую же рамку, как и в примере выше.

Несколько самых распространенных стилей рамок:

* solid — сплошная;
* dashed — пунктирная;
* dotted — точками.

Рамку можно задавать и строчным, и блочным элементам.

## Стандартная блочная модель [7/23]

Стандартная блочная модель отвечает на основной вопрос:

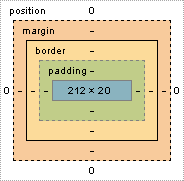
Сколько же в итоге места будет занимать элемент?

Ответ следующий:

Область, занимаемая *блочным* элементом, складывается из его ширины и высоты содержания, внутренних и внешних отступов, ширины рамок .

Для строчных элементов есть свои особенности, которые мы опишем позже.

Схема блочной модели:



* ширина/высота содержания — свойства width и height (синий прямоугольник на схеме);
* внутренние отступы — свойство padding;
* рамки — свойство border;
* внешние отступы — свойство margin.

## Схлопывание» внешних отступов [9/23]

Во второй части курса разберем некоторые важные тонкости блочной модели. И начнем с внешних отступов (или маргинов).

Вертикальный отступ между двумя соседними элементами равен максимальному отступу между ними. Если отступ одного элемента равен 20px, а второго 40px, то отступ между ними будет 40px.

Этот эффект называется эффектом «схлопывания» внешних отступов или «схлопывания» маргинов.

Горизонтальные отступы между элементами просто складываются. Например, горизонтальный отступ между двумя элементами с отступами 30px будет равен 60px.

## «Выпадание» внешних отступов [10/23]

«Выпадание» — это еще один эффект, связанный с вертикальными внешними отступами. Если внутри родительского блока расположить блок и задать ему отступ сверху, то внутренний блок прижмется к верхнему краю родительского, а у родительского элемента появится отступ сверху. Т.е. верхний отступ внутреннего элемента «выпадает» из родительского элемента.

Если у родительского элемента тоже был задан внешний отступ, то выберется максимальный отступ между собственным и «выпавшим» .

Чтобы избавиться от эффекта выпадания, можно задать родительскому элементу внутренний отступ (паддинг) сверху или добавить рамку сверху.

Внешние и внутренние отступы всегда складываются.

## Как отцентровать элемент? [11/23]

Чтобы отцентровать *блочный* элемент, нужно выполнить следующие действия:

1. Задать элементу ширину, которая меньше ширины родительского контейнера.
2. Задать для внешних отступов справа и слева значение auto.

Примеры кода:

*selector* {

width: 100px;

margin: 0 auto;

}

*selector* {

width: 100px;

margin-left: auto;

margin-right: auto;

}

## Блочная модель и строчные элементы [12/23]

Еще раз акцентируем внимание на особенности поведения строчных элементов в блочной модели:

1. Не реагируют на CSS-свойства width и height.
2. Частично реагируют на margin, воспринимая только горизонтальные отступы.
3. Частично реагируют на padding, воспринимая только горизонтальные отступы.
4. При задании вертикальных padding визуально увеличиваются, но без увеличения занимаемого места (не отталкивают другие элементы).
5. Воспринимают рамки. Аналогично padding рамки сверху и снизу не увеличивают занимаемое элементом место.

После выполнения задания обратите внимание на то, что расстояние между строк в абзацах не увеличилось, хотя элементы визуально стали выше.

## Ширина 100% и ширина по умолчанию [13/23]

CSS-свойство width задаёт не общую ширину блока, а только ширину содержания. Общая ширина блока затем складывается из трёх компонентов: ширины содержания, внутренних отступов и ширины рамок слева и справа.

Поведение элемента может зависеть от того, как именно вы зададите его ширину.

**Первый вариант**. Вариант по умолчанию, когда ширина не задается, соответствует значению width:auto;. В этом случае блок занимает всю ширину родительского блока. Если у блока есть внутренние отступы или рамки, то его ширина содержания автоматически уменьшается, а общая ширина остается равной ширине родителя.

**Второй вариант**. Когда ширина блока задана явно, например, width:100%;. В этом случае ширина содержания блока равна ширине родительского блока. Если блоку добавить внутренние отступы и рамки, то его общая ширина становится больше ширины родителя.

В этом задании вы увидите, как эти эффекты работают на практике. Обратите внимание, что ширины блоков будут рассчитываться автоматически.

## Проблемы обычной блочной модели [14/23]

В чём проблема описанного механизма расчета ширины?. Ведь можно просто не задавать ширину, используя auto, либо просто подбирать размеры блоков и отступов как на макете.

Проблемы начинаются, когда сетка резиновая и элементы в ней должны тянуться. Самый простой пример: форма, в которой поля должны занимать всю ширину контейнера, но при этом иметь фиксированные внутренние отступы, чтобы текст не прилипал к краям.

Для такой формы можно задать такие стили:

input[type="text"] {

width: 100%;

padding: 0 10px;

}

Ширина 100% заставит поля быть такой же ширины, как родительский блок, а паддинг добавит отступы. Но, как вы уже догадались, за счет отступов поле станет шире своего контейнера. Попрактикуемся.

Ширина полей ввода определяется значением атрибута size. Ширина width: auto для полей ввода рассчитывается из значения size по умолчанию и не растягивает поля на всю ширину контейнера.

## Изменяем блочную модель, свойство box-sizing [15/23]

Проблему с шириной тянущихся полей решить сложно. Идеальным вариантом было бы изменение алгоритма расчета ширины элемента, чтобы свойство width задавало не *ширину содержания*, а *общую ширину*.

К счастью, такая возможность была добавлена в CSS3 с помощью свойства box-sizing, которое уже поддерживается большинством современных браузеров.

Это свойство имеет два значения:

1. content-box — значение по умолчанию, соответствует стандартной блочной модели.
2. border-box — изменяет режим расчета ширины элемента на описанный выше.

Полезная статья про box-sizing на русском: [Используйте свойство box-sizing](http://simonenko.su/32197993404/use-css3-box-sizing)

## Управление типом элемента, свойство display [17/23]

Тип элемента не является чем-то вечным и неизменным, его можно изменять с помощью CSS. За это отвечает свойство display.

С его помощью, например, можно сделать абзацы и заголовки строчными, а спаны и стронги блочными элементами.

У свойства display много значений. С двумя из них вы уже заочно познакомились. Свойство display:block обозначает блочный элемент, а display:inline — строчный.

В третьей части курса мы познакомимся с некоторыми более продвинутыми значениями свойства display.

## display: inline-block [18/23]

Иногда возникает необходимость расположить в ряд несколько элементов с заданными размерами. Строчные элементы для этого не подходят, т.к. не воспринимают размеры. Блочные элементы тоже не подходят, т.к. до и после них существует перенос строки. Конечно, блочные элементы можно приспособить для такой задачи, используя дополнительные свойства (которые будут разбираться в курсе про сетки).

Но более простой способ — использовать *блочно-строчные* элементы. В HTML нет тегов, которые по умолчанию вели бы себя как блочно-строчные, но любой элемент можно переключить в данный режим, задав ему свойство display со значением inline-block.

Особенности блочно-строчных элементов:

* им можно задавать размеры, рамки и отступы, как и блочным элементам;
* их ширина по умолчанию зависит от содержания, а не растягивается на всю ширину контейнера;
* они не порождают принудительных переносов строк, поэтому могут располагаться на одной строке, пока помещаются в родительский контейнер;
* элементы в одной строке выравниваются вертикально подобно строчным элементам.

## display: table [19/23]

Дальнейшие несколько заданий будут посвящены табличным типам элементов.

Табличные сетки были очень популярны на заре веба. Действительно, у табличных сеток есть преимущества, которые очень сложно получить с помощью блоков. Например, столбцы одинаковой высоты.

Однако, семантически таблицы не предназначены для разметки сеток и постепенно от них отказались. Но память об их удобстве жила.

Позднее было разработано семейство значений свойства display, которое позволяет задать табличное поведение любым элементам.

Первое значение — display:table задает элементу тип *таблица*. Особенности табличных элементов:

1. можно задавать ширину, высоту, рамки, отступы;
2. по умолчанию ширина зависит от содержания;
3. переносы строки до и после элемента.

Табличные элементы похожи на блочные за исключением ширины по умолчанию.

## display: table-row [20/23]

Как вы догадались, мы строим таблицу на дивах с помощью новых значений свойства display. Знания про таблицы вы можете освежить в курсе [Знакомство с таблицами](https://htmlacademy.ru/courses/39).

Любая таблица содержит элементы строка таблицы, внутри которых должны содержаться элементы ячейка таблицы. Соответствие тегов таблицы значениям display:

<table> — display:table;

<tr> — display:table-row;

<td> — display:table-cell;

В этом задании вы создадите строки таблицы с помощью display:table-row;. Строка — необычный элемент. Она является контейнером для ячеек и практически не имеет собственного отображения. Для нее можно только задавать цвет фона.

## display: table-cell [21/23]

С помощью значения table-cell свойства display можно задать элементу тип ячейка таблицы. Конечно, лучше чтобы элементы-ячейки находились внутри элементов-строк, которые находятся внутри элементов-таблиц.

Вы можете просто задать элементу тип ячейка таблицы, не добавляя вокруг него дополнительных элементов-строк и таблиц. В этом случае браузер создаст дополнительные анонимные элементы строки и таблицы. Неудобство заключается в том, что вы не сможете ими управлять.

К таблицам, созданным с помощью CSS, можно применять те же свойства, что и к обычным таблицам. Например, задавать отступы между ячейками или режим схлопывания границ.

## display: none [22/23]

Значение none свойства display используется очень часто. С его помощью можно скрыть элемент, как будто его и не было. Скрытый элемент не отображается и не занимает места на странице.

Данное свойство применяется при создании выпадающих меню, динамических галерей, переключающихся вкладок и много где еще.

Есть еще одно CSS-свойство, которое используется для сокрытия элементов. Это свойство visibility со значением hidden. Оно «прячет» элемент — он становится невидимым, но занимает место на странице.

# Сетки

## Пробуем управлять потоком [1/32]

О понятии [потока документа](https://htmlacademy.ru/courses/45/run/1) рассказывается в курсе про позиционирование.

Поток — это порядок отображения элементов на странице. По умолчанию блочные элементы отображаются как прямоугольные области, идущие друг за другом сверху вниз, а строчные элементы располагаются сверху вниз и слева направо и при необходимости переносятся на новую строку.

Потоком можно управлять и изменять привычное поведение элементов в потоке. Например, можно заставить блочные элементы двигаться не сверху вниз, а выстраиваться в несколько колонок.

А научившись управлять потоком, вы сможете строить сетки.

В первой серии заданий мы продемонстрируем как один и тот же HTML-код можно выстроить в разные потоки. Например, вот так:



Вам нужно просто выполнять указания системы.

## Управление потоком, шаг 2 [2/32]

На первом шаге мы с помощью свойства float изменили поведение первого блока.

Обратите внимание, как он стал наезжать на последующие блоки. Это одна из особенностей флоатов, которую мы разберём позже в курсе.

А пока продолжим: сейчас будем изменять поведение второго блока.

## Управление потоком, шаг 3 [3/32]

Со вторым блоком разобрались. Теперь уже первые два блока наезжают на третий.

Чтобы получить нужный нам поток, осталось задать размеры третьему блоку, отцентровать его и избавиться от эффекта «наезжания».

Поможет нам в этом свойство clear.

## Создадим другой поток [4/32]

У вас получилось создать первый поток. Теперь попробуем те же самые элементы выстроить по-другому. Вот так:



Просто выполняйте указания системы.

## Другой поток, шаг 2 [5/32]

А пока вы пошагово создаёте второй поток, поговорим о сетках.

Сетка — это взаимное расположение крупных блоков сайта. Вспомните любой сайт, например, ВКонтакте или HTML Academy.

У ВКонтакте простая сетка, которая состоит из верхней части, или «хедера», под которой располагаются две колонки, а в самом низу расположен «футер» или нижняя часть сайта. Вот так схематично можно представить сетку ВКонтакте:

-------------------------------------

хедер

-------------------------------------

левая колонка | правая колонка

меню | основное содержание

-------------------------------------

футер

-------------------------------------

У HTML Academy на разных страницах используются разные сетки. Например, на главной странице есть двухколончатый хедер, под ним содержание страницы в три колонки, затем нижнее меню и двухколончатый футер.

Как вы понимаете, сетки могут быть самыми разными.

## Другой поток, шаг 3 [6/32]

Что нужно знать и уметь, чтобы строить сетки? Две вещи:

1. управлять размерами элементов;
2. управлять поведением элементов в потоке.

Первая часть детально разбирается в курсе «[Блочная модель документа](https://htmlacademy.ru/courses/44)». Там вы учитесь:

* задавать размеры;
* рамки;
* внешние и внутренние отступы;
* рассчитывать полный размер элемента;
* рассчитывать занимаемую им площадь.

Управлять поведением в потоке мы будем учиться в этом курсе.

## Другой поток, финал [7/32]

Существует несколько способов управлять потоком и строить сетки:

1. флоаты;
2. инлайн-блоки;
3. табличная вёрстка;
4. флексбоксы.

Табличная вёрстка — самый простой для понимания способ построения сеток. Но он считаетсяустаревшим и использовать его не рекомендуется. Сейчас уже можно использовать для такой вёрстки не таблицы, а элементы с display: table, display: table-row и display: table-cell. Подробно эта тема рассматриваются в курсе [«Таблицы на CSS»](https://htmlacademy.ru/courses/86).

Флексбоксы — это новая и очень мощная технология для построения сеток. Подробно она разбирается в курсах [Флексбокс, часть 1](https://htmlacademy.ru/courses/96) и [Флексбокс, часть 2](https://htmlacademy.ru/courses/113).

В этом курсе мы подробно разберём флоаты и инлайн-блоки. Основной упор сделаем на флоаты, так как сейчас на многих сайтах сетки построены на них.

А пока что завершите создание своей второй сетки.

## Погружение в флоаты [8/32]

А теперь детально разберём свойство float, его значения, особенности поведения и способы применения.

Т. к. у этого свойства нет устоявшегося русского перевода или термина, то мы будем называть его «флоатом», а также использовать конструкции наподобие «зафлоатить».

Изначально float было предназначено для того, чтобы включать обтекание элементов текстом. Наподобие того, как в более старой версии HTML текст обтекал изображения c атрибутами align="left" или align="right".

То есть, можно сказать, что float — это свойство, включающее режим обтекания. Но, как часто бывает, судьба уготовила ему совсем другую роль.

Свойство float имеет следующие значения:

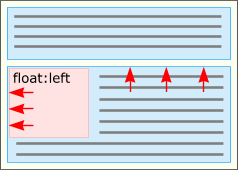
1. left — прижимает элемент к левому краю родителя, другие элементы обтекают его справа;
2. right — прижимает элемент к правому краю родителя, другие элементы обтекают его слева;
3. none — отключает режим обтекания и возвращает элементу нормальное поведение.

Обратите внимание, что зафлоатить элемент по центру нельзя.

Иллюстрация в задании взята из статьи [«Раскладка в CSS: float»](http://softwaremaniacs.org/blog/2005/12/01/css-layout-float/).

Как и позиционирование, float используется для того, чтобы двигать боксы.

Но в отличие от позиционирования, которым можно двигать боксы практически произвольно, все, что может float — это сдвинуть элемент к одной из сторон потока, правой или левой.

 При этом сам бокс и следующие за ним в потоке приобретают интересное поведение: float'нутый бокс смещается по горизонтали и прилипает к одной из сторон родителя.

## float и ширина [9/32]

Вы уже знаете, что по умолчанию блочные элементы растягиваются на всю доступную ширину родителя.

Если мы задаём элементу свойство float:left или float:right, то он прижимается к левому или правому краю, а также начинает ужиматься по ширине под своё содержимое. С той стороны, которая не прижата к краю родителя, появляется свободное место. Это место может быть занято другими элементами.

Зафлоаченному элементу можно явно задавать размеры и отступы.

Есть тонкость, связанная со строчными элементами. Если зафлоатить строчный элемент, то он начинает вести себя как блочный, а именно: воспринимать размеры и отступы.

## float и выпадание из потока [10/32]

Про выпадание элементов из потока при абсолютном и фиксированном позиционировании мы поговорим в курсе [«Позиционирование»](https://htmlacademy.ru/courses/45). А сейчас рассмотрим выпадание при флоатах.

Зафлоаченные элементы выпадают из потока, но лишь частично:

* Блочные элементы, которые идут в коде после зафлоаченного блока, перестают его замечать. Они подтягиваются вверх и занимают его место, как будто его и нет.
* Строчные же элементы, расположенные в коде после зафлоаченного блока, начинают обтекать его со свободной стороны.

Ещё раз: для блочных элементов флоатные не существуют, но текст внутри блоков флоатные обтекает.

Такое поведение флоатов даёт интересные эффекты:

1. Эффект прохождения сквозь блоки. Проявляется, когда зафлоаченный элемент выше, чем несколько последующих за ним блоков.
2. Эффект выпадания из родителя или схлопывания родителя. Проявляется тогда, когда все дочерние блоки в родителе зафлоачены. В этом случае родитель схлопывается по высоте, как будто в нём нет содержимого, а блоки выпадают из него.

## Флоат рядом с флоатом [11/32]

Флоатные элементы становятся невидимыми для блочных элементов и видимыми для текста. А как же они взаимодействуют друг с другом?

Если вкратце то: флоатные элементы видят друг друга.

Идущие друг за другом флоаты выстраиваются в ряд, пока им хватает свободного места. Если места не хватает, то они начинают переноситься на следующую строчку. Почти как текст.

Как раз эта особенность флоатов и позволила применять их для создания сеток. Ведь в начале двухтысячных в CSS не было никакого другого способа создавать колонки и задавать им размеры.

Блочные элементы всегда располагались в разных строках. А строчные элементы, хоть и могли располагаться на одной строке, совершенно не воспринимали размеры. Позиционирование тоже не подходило для сеток, т. к. элементы выпадали из потока.

В общем, флоаты оказались как нельзя кстати для тех, кто решил уходить с табличной вёрстки.

## Когда флоатов много, а места мало [12/32]

Следует отметить, что поведение нескольких флоатов, когда им не хватает места в одной строке, является очень странным.

Когда не влезающий флоат переносится на новую строку, возможно несколько вариантов и не все из них логичны. Например, флоат может «зацепиться» за один из предшествующих флоатов и встать ниже не в самом начале строки, а за предшествующим.

В этом упражнении примеры специально подобраны так, чтобы вы увидели странности поведения флоатов при недостатке места.

Естественно, эти особенности повлияли на приёмы использования флоатов в создании раскладок (еще одно название для сеток). Но об этом позже.

## Свойство clear [14/32]

Свойство clear запрещает обтекание элемента другими элементами. Вот его значения:

1. left — запрещено обтекание слева;
2. right — запрещено обтекание справа;
3. both — запрещено обтекание с обеих сторон;
4. none — обтекание разрешено.

Если после флоатного элемента расположен элемент с запрещённым обтеканием, то последний опускается под флоатный.

Свойство clear учит блочные элементы «видеть» зафлоаченные.

## Борьба с выпаданием флоатов: распорки [15/32]

Эффект выпадания флоатов из родителя был большой проблемой при построении сеток. Ведь сетки на флоатах обычно делаются вот так:

1. Создаём блок-контейнер для колонок.
2. В контейнер добавляем флоатные блоки-колонки.
3. Рассчитываем ширины колонок так, чтобы им хватало места в родителе.

Такой подход работает неплохо. Но проблемы начинаются тогда, когда у контейнера есть фон, отличный от фона страницы. Когда колонки выпадали, родитель схлопывался и фон пропадал.

Необходимо было найти способ борьбы с выпаданием. Тут и пригодилось свойство clear:both. Стали применять такую структуру:

<div class="container"> - блок-контейнер

<div class="column1">...</div> - колонка, флоат

<div class="column2">...</div> - колонка, флоат

<div class="clearfix"></div> - распорка с clear:both

</div>

Т. е. начали добавлять после зафлоаченных колонок пустой элемент-распорку со свойством clear:both. Этот элемент видел колонки, не давал им пройти через себя, а заодно и растягивал родительский блок по высоте.

Для таких распорок прижилось специальное название класса — clearfix.

## Борьба с выпаданием флоатов: псевдораспорки [16/32]

Хорошие верстальщики не очень любят, когда в вёрстке появляются ненужные дополнительные элементы и стараются от них избавляться.

До поры до времени с блоками-распорками приходилось мириться, т. к. без них было никак не обойтись. Но время шло и в браузерах появилась поддержка так называемых псевдоэлементов.

Псевдоэлементы позволяют с помощью CSS вставить в структуру HTML-документа узлы, которых нет в HTML коде. Т. е. можно вставить в код элемент и не писать его в HTML. Благодаря псевдоэлементам появилось решение проблемы флоатов без использования дополнительного элемента.

Назовём это решение псевдораспорками. Есть несколько его вариаций, но вот одна из распространённых. В CSS добавляется следующее правило:

.clearfix::after {

content: "";

display: table;

clear: both;

}

А затем класс clearfix добавляется к контейнеру, внутри которого лежат флоатные колонки. После этого в контейнер не нужно добавлять дополнительный элемент-распорку, т. к. распорка создаётся с помощью псевдоэлемента.

Подробнее псевдоэлементы мы разберем в одном из последующих курсов.

## Простейшая сетка, шаг 1 [17/32]

Мы разобрали по косточкам свойство float, познакомились с clear и научились бороться с некоторыми проблемами флоатов.

В этой части курса мы будем практиковаться строить сетки на флоатах.

И начнём с простейшей сетки «хедер - две колонки - футер». Одна колонка будет фиксированной ширины, а вторая будет растягиваться в зависимости от ширины контейнера.

В исходном состоянии у нас 4 блока, расположенных друг под другом. Первым делом займёмся боковой панелью или «сайдбаром»: зафлоатим её влево и зададим её фиксированную ширину.

## Простейшая сетка, шаг 2 [18/32]

Мы зафлоатили сайдбар, и он вывалился из контейнера и залез на футер. Исправим это с помощью псевдораспорки.

Вам нужно будет добавить контейнеру колонок класс clearfix. А после этого добавить несколько свойств в CSS-правило для псевдораспорки.

1. **цель 1**Добавьте элементу с классом content-container класс clearfix.
2. **Цель 2**Добавьте CSS-правилу .clearfix::after свойство content со значением "".
3. **Цель 3**Туда же добавьте свойство display со значением table
4. **Цель 4**и свойство clear со значением both.

## Простейшая сетка, финал [19/32]

Поздравляем! Вы победили выпадание. Теперь завершим сетку.

В почти готовую сетку мы добавили немного текста, чтобы посмотреть как она будет себя вести с наполнением. Сейчас видно, что синий блок контента частично залезает под сайдбар. Чтобы избавиться от этого эффекта, можно задать блоку контента маргин слева.

После этого сетка будет завершена. Кстати, мы получили ещё один интересный результат. Мы не зафлоатили блок контента, поэтому он тянется на всю ширину родительского контейнера. А фиксированный маргин слева предохраняет его от заползания под сайдбар.

Таким образом, у нас получилась сетка из двух колонок, одна из которых фиксированная, а вторая тянется. Здесь важно следующее — сайдбар в HTML коде должен располагаться до блока контента, иначе такого эффекта не получится.

## Cетка посложнее, шаг 1 [20/32]

Теперь построим более сложную сетку. В конце у нас получится вот такой результат:



Да, мы повторяем раскладку прошлой версии главной страницы HTML Academy: двухколоночный хедер и контент в три колонки.

На этом шаге создадим сетку хедера. Пока что работаем без содержания.

## Сетка посложнее, шаг 2 [21/32]

Обратите внимание на способ именования классов: колонки имеют класс layout-column.

Префикс layout обозначает «сетка» или «раскладка». Таким образом, мы даём понять, что классы с этим префиксом предназначены для создания сетки. Обычно для таких классов задают только флоаты, размеры и внешние отступы.

Использование значимых префиксов для именования классов — хороший приём. Используйте его в своей вёрстке.

На втором шаге мы создадим колонки содержания.

## Сетка посложнее, шаг 3 [22/32]

Теперь нам нужно задать отступы между колонками содержания. Сделать это просто — добавим им маргин справа.

Но есть небольшая проблема. По макету последняя колонка должна прижиматься к правому краю контейнера. Поэтому нам нужно обнулить маргин справа у последней колонки в содержании, иначе она не будет влезать в контейнер и перенесётся на следующую строку.

Чтобы выбрать последнюю колонку, вы можете использовать псевдокласс :last-child, с которым вы [уже знакомились](https://htmlacademy.ru/courses/42/run/9) в курсе про селекторы.

## Сетка посложнее, добавляем содержание [23/32]

Сетка готова. Осталось протестировать её с настоящим содержанием.

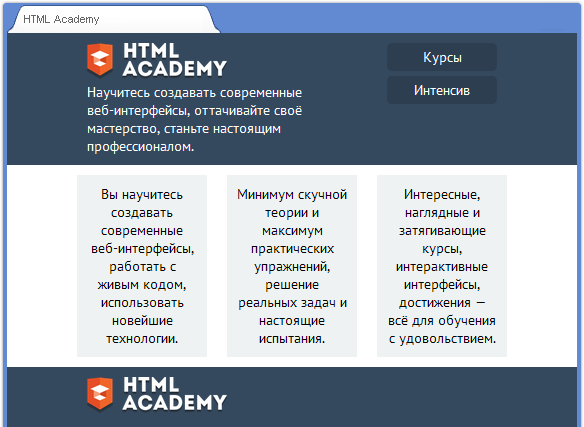
Мы уже добавили контент в разметку и спрятали его. Вам нужно лишь снова отобразить его, а также убрать вспомогательные границы и фоны, чтобы увидеть готовый результат.

Обратите внимание еще на один подход построения сеток. Он заключается в том, что блокам разметки задаются только размеры и внешние отступы. Внутренние же отступы добавляются дополнительным элементам, вложенным в ячейки сетки.

Такой подход упрощает расчет сеток. Как вы помните, блочная модель устроена таким образом, что общий размер элемента складывается из ширины/высоты, маргинов и паддингов, а также ширины рамок. Поэтому, чтобы получить правильные размеры блоков сетки, всегда нужно помнить об этом. Если же блокам сетки задавать только ширину и внешние отступы, то слагаемых становится намного меньше, а шанс совершить ошибку уменьшается.

## Последняя сетка, шаг 1 [24/32]

Поздравляем с созданием первой сложной сетки! Теперь давайте немного усложним её. Мы будем делать сетку, у которой фон хедера и футера растягивается на всю ширину окна браузера, вот так:



Сначала нам нужно изменить HTML-разметку. Мы избавимся от блока wrapper, который центрует всё содержание страницы.

Затем нужно будет добавить внутрь хедера, контента и футера дополнительный блок, который будет центровать их содержимое. Вот так:

<div class="header clearfix">

<div class="layout-positioner">

<!-- блоки колонок -->

</div>

</div>

## Последняя сетка, шаг 2 [25/32]

Блоки с классом layout-positioner добавлены в код. Это невидимые блоки-центровщики. Теперь хедер и футер могут тянуться на всю ширину окна, а их содержимое будет отцентровано с помощью добавленных блоков.

Мы задали голубую рамку для центровщиков, чтобы было видно, как они себя ведут. Также мы добавили им псевдораспорки, поэтому можно удалить класс clearfix из HTML кода.

И еще одна мелочь: класс wrapper уже нигде не используется, поэтому он удалён из CSS. Класс clearfix мы оставили в CSS, т. к. это вспомогательный класс, который может пригодиться при вёрстке.

На этом шаге ваша задача — отцентровать центровщики.

## Последняя сетка завершена [26/32]

Последний шаг этой серии заданий — тестирование получившейся сетки с содержанием.

Контент уже добавлен и скрыт. Снова отобразите его, а также уберите вспомогательные рамки и фоны.

Приём с дополнительными блоками, которые позиционируют содержание, чтобы родительские блоки могли тянуться на всю ширину окна, достаточно распространён. Дизайн с подобными сетками тоже встречается очень часто.

.layout-positioner {

width: 430px;

margin: 0 auto;

}

.layout-positioner::after {

display: table;

content: "";

clear: both;

}

## Погружение в inline-block [28/32]

C блочно-строчными элементами вы [уже знакомились](https://htmlacademy.ru/courses/44/run/18) в курсе про блочную модель документа. Напомним, что это элементы, у которых свойство display имеет значение inline-block.

Блочно-строчные элементы ведут себя двояко. Снаружи они выглядят как обычные строчные, но внутри они ведут себя как блочные.

От строчных им достались следующие черты:

* по ширине они ужимаются под своё содержимое;
* могут располагаться в одну строку;
* реагируют на вертикальное выравнивание, vertical-align;
* реагируют на горизонтальное выравнивание, text-align, заданное у родителя.

От блочных:

* им можно задавать размеры с помощью width и height;
* а также внешние и внутренние отступы и рамки, которые работают во всех направлениях и увеличивают размер элемента.

## float vs inline-block [29/32]

Блочно-строчные элементы могут выстраиваться друг за другом, а если не хватает места, то они переносятся на следующую строку.

Поведение похоже на флоатные элементы, однако, есть существенное отличие.

При переносе на следующую строку блочно-строчные ведут себя намного логичней. Высота строк у них рассчитывается по максимальному элементу с учётом отступов.

Это упражнение создано, чтобы продемонстрировать отличие в поведении флоатов и инлайн-блоков. В исходном состоянии мы видим несколько флоатных блоков, которые не смогли выстроиться в ряды, так как зацепились друг за друга.

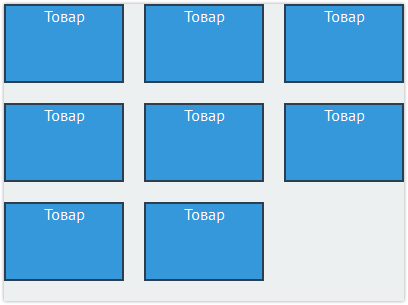
Вы измените тип блоков на блочно-строчные и сами всё увидите, насколько умнее ведут себя инлайн-блоки.

## Простая сетка на inline-block [30/32]

Блочно-строчные элементы очень часто используют для создания декоративных элементов: кнопок, плашек, блочков. Также благодаря их умному поведению с их помощью создают различные списки товаров в каталогах.

Для создания сеток страниц эти элементы используются реже. Первая причина заключается в том, что их не поддерживают старые браузеры, например, IE7 и младше. А сетка страницы — слишком критичная вещь, чтобы позволять ей ломаться даже в старых браузерах.

Есть и еще одна проблема, с которой мы столкнёмся в этом задании. Давайте попробуем построить сетку, как на картинке, когда у нас края товаров вплотную прилегают к краям контейнера:



Ширина контейнера 400px, ширина товара с рамками 120px, отступ между товарами 20px. Чтобы три товара влезали в строку «пиксель-в-пиксель», нужно у каждого третьего товара обнулить отступ справа. Используем для этого селектор .item:nth-child(3n).

## inline-block и пробелы в коде [31/32]

Мы рассчитали всё правильно, однако по три товара в строку не помещается.

Причина заключается в пробелах после тэгов в HTML-коде. Блочно-строчные ведут себя как текст, поэтому если в коде есть пробел между элементами, то он отображается и на странице. Этот пробел увеличивает отступы между товарами, не давая им поместиться в одну строку.

Бороться с пробелом после блочно-строчных можно несколькими способами:

* удалять пробелы в коде;
* обнулять размер шрифта;
* играться с маргинами после блочно-строчного.

У каждого из способов есть свои недостатки, а подробнее эти и другие способы разбираются в [нашем переводе хорошей статьи](https://htmlacademy.ru/blog/21) (есть ещё более обширная [статья](http://css-live.ru/articles/zagadochnye-otstupy-mezhdu-inlajn-blokami.html) про блочно-строчные).

Мы попробуем последние два способа.

Способ со шрифтом заключается в том, что мы задаём нулевой размер шрифта у контейнера инлайн-блоков, а самим инлайн-блокам задаём исходный размер шрифта. Способ не работает, если вы используете относительные размеры шрифта.

Способ с маргинами заключается в том, что мы уменьшаем отступ после инлайн-блока на ширину пробела, около 4-5px. А если нам нужно, чтобы элементы стояли вплотную друг к другу, то задаём отрицательный отступ. Проблема с этим способом заключается в том, что размер пробела может быть разным в разных шрифтах и может изменяться при изменении размера шрифта.

# Позиционирование

## Поток документа [1/20]

Порядок отображения элементов на странице называется потоком документа. Блочные элементы отображаются как прямоугольные области, идущие друг за другом сверху вниз, а строчные элементы располагаются сверху вниз и слева направо и при необходимости переносятся на новую строку.

Элементы можно вкладывать друг в друга. Чем раньше в коде расположен элемент, тем выше он расположен на странице.

Некоторые типы элементов могут вести себя в потоке более сложно. Например, они могут располагаться в несколько колонок, как блочно-строчные элементы. Эти типы проходились в курсе «[Блочная модель документа](https://htmlacademy.ru/courses/44)».

Любой элемент, который находится в потоке, занимает на странице свою собственную площадь и, если надо, отталкивает от себя соседей с помощью маргинов.

В этом курсе мы будем изучать свойства, которые изменяют обычное поведение элементов в потоке.

## Относительное позиционирование [2/20]

CSS-свойство position задает режим позиционирования элементов. Значением по умолчанию является static, которое означает «обычное позиционирование».

Значение relative обозначает «относительное позиционирование». Относительно спозиционированный элемент можно перемещать относительно его исходного положения.

Относительно спозиционированный элемент остается в потоке документа, а при его перемещении наблюдаются интересные эффекты. Для начала просто зададим относительное позиционирование выбранному блоку.

## position: relative и свойство top [3/20]

Пока что ничего не произошло, но мы еще и не перемещали элемент. Чтобы эффект перемещения был более наглядным, блок .block-relative обёрнут в блок .block-shadow, который служит «тенью» элемента.

Блок-тень имеет те же размеры, что и блок внутри него. Это позволит нам отследить, как изменяется поведение относительно спозиционированного элемента в потоке.

Свойство top позволяет сместить относительно спозиционированный элемент вверх или вниз относительно его исходного положения. Значение этого свойства обычно задается в пикселях, но можно использовать и другие единицы измерения CSS.

## position: relative и свойство left [4/20]

Вы заметили, что когда мы сместили блок вниз, нижележащие блоки не сдвинулись?

Это происходит потому, что относительно спозиционированный элемент на самом деле никуда не смещается. Он остаётся в потоке ровно там же, где и был (это место обозначено блоком-тенью). А смещается иллюзорная копия блока.

Свойство left позволяет сместить блок влево или вправо относительно исходного положения.

В этом задании вы не только сместите блок по горизонтали, но и убедитесь в том, что он остался в потоке, увеличив его высоту. При этом нижний блок должен будет сместиться.

## position: relative и свойство bottom [5/20]

Свойство bottom аналогично свойству top, только смещает элемент в другом направлении.

Положительное значения свойства bottom сместит элемент вверх, а отрицательное вниз.

Немного попрактикуемся.

## position: relative и свойство right [6/20]

Свойство right аналогично свойству left и тоже смещает элемент в другом направлении.

Как вы уже убедились, относительно спозиционированный элемент можно смещать по горизонтали и по вертикали, комбинируя свойства top/bottom со свойствами left/right.

Ещё раз убедимся, что элемент на самом деле никуда не смещается, увеличив отступ снизу. Размер отступа будет расчитываться от нижнего края блока-тени, а не от края смещенного блока-иллюзии.

## Относительное позиционирование на практике [7/20]

В реальных проектах относительное позиционирование может использоваться для создания декоративных эффектов, хотя область его применения намного шире.

В этом задании вы создадите псевдотень для изображений. Все стили уже подготовлены, не хватает только тех, которые отвечают за позиционирование.

## Абсолютное позиционирование [8/20]

Значение absolute свойства position задаёт элементу абсолютное позиционирование.

Абсолютно спозиционированные элементы обладают следующими особенностями:

1. Выпадают из потока документа. Место, которое они занимали, становится как бы пустым и его занимают соседние элементы.
2. Ширина по умолчанию зависит от содержимого (а не растягивается на всю доступную ширину).
3. Остаются на том же месте, где были, если не заданы значения свойств top, left, right, bottom.

## Абсолютное позиционирование и строчные элементы [9/20]

Абсолютное позиционирование изменяет поведение не только блочных элементов (ширина по умолчанию), но и строчных.

Если строчный элемент спозиционирован абсолютно, то он ведет себя точно так же, как и абсолютно спозиционированный блочный. Например, такому строчному элементу можно задавать размеры с помощью width и height.

Можно сказать, что есть еще один тип элементов — абсолютно спозиционированные. Такими становятся элементы всех остальных типов (блочные, строчные, блочно-строчные и т.д.), если им задано свойство position: absolute;.

## position: absolute и свойство left [10/20]

Свойства left, top, right и bottom по-разному работают для элементов с абсолютным и относительным позиционированием.

Если для «относительных» элементов эти свойства задают смещение относительно исходной позиции, то для «абсолютных» элементов они задают расположение относительно некой системы координат, как для точки на графике.

По умолчанию такой системой координат является окно браузера, а точки начала координат привязаны к его углам.

Свойство left задает отступ от *левого* края окна браузера до *левого* края абсолютно спозиционированного элемента. Если свойству задать отрицательное значение, то элемент сместится за левый край браузера (при этом полоса прокрутки не появится).

## position: absolute и свойство top [11/20]

Свойство top задает отступ от *верхнего* края окна браузера до *верхнего* края абсолютно спозиционированного элемента.

Таким образом, с помощью свойств left и top можно позиционировать «абсолютный» элемент относительно левого верхнего угла окна браузера.

## position: absolute и свойство right [12/20]

Свойство right задает отступ от *правого* края окна браузера до *правого* края «абсолютного» элемента.

Это свойство позволяет легко прижать элемент любой ширины к правому краю окна. Если задать отрицательное значение для right, то элемент сместится за край окна, но при этом появится полоса прокрутки.

C помощью top + right можно позиционировать элементы от правого верхнего угла.

## position: absolute и свойство bottom [13/20]

Свойство bottom задает отступ от *нижнего* края окна браузера до *нижнего* края «абсолютного» элемента.

В связи с особенностями нашего мини-браузера к свойству bottom добавляется дополнительный отступ, которого в обычных браузерах нет.

Отрицательное значение bottom затолкнет элемент за нижний край окна браузера и при этом появится полоса прокрутки.

## Точка отсчёта координат [14/20]

По умолчанию «абсолютные» элементы позиционируются в системе координат, привязанной к окну браузера. Однако, систему координат можно изменить.

Если какой-нибудь из родителей «абсолютного» элемента имеет относительное позиционирование, то элемент будет позиционироваться относительно этого родителя. Если среди родителей несколько относительно позиционированных элементов, то «абсолютный» элемент будет позиционироваться относительно ближайшего из них.

В этом задании вы увидите, как происходит смена системы координат (и соответственно точек отсчета). Интересно, что позиционирование по умолчанию отличается от позиционирования внутри тега body.

## Тренируемся задавать координаты [15/20]

В этом задании вы потренируетесь в позиционировании. Задача заключается в том, чтобы с помощью свойств left, top, right и bottom расположить красный квадрат в каждом из четырех углов синего прямоугольника.

Синие прямоугольники спозиционированы относительно, поэтому у каждого красного квадрата своя система координат.

## Неявная точка отсчёта [16/20]

Неявная точка отсчета — это то место, где находился элемент до того, как ему задали абсолютное позиционирование. Если «абсолютному» элементу не задавать свойства left ... bottom, то он выпадет из потока, но останется на прежнем месте.

Такой элемент можно сместить из неявной точки отсчета с помощью свойства margin. Оно будет работать примерно так же, как и свойства left ... bottom для «относительного» элемента.

Кстати, когда значение для left не задано, оно равно auto. То же верно и для оставшихся трёх свойств. С помощью значения auto можно возвращать «абсолютные» элементы на исходное положение.

## Абсолютное позиционирование на практике [17/20]

На практике абсолютное позиционирование часто применяют в декоративных целях, когда нужно аккуратно разместить какие-либо иконки, управляющие элементы или другие «рюшечки».

В этом задании вы узнаете, как применяется абсолютное позиционирование в блоках достижений в HTML Academy. С помощью внутреннего отступа в левую часть блока добавляется пустое пространство, в которое с помощью абсолютного позиционирования выносится иконка достижения.

## Фиксированное позиционирование [18/20]

Значение fixed свойства position задаёт элементу фиксированное позиционирование.

Фиксированное позиционирование похоже на абсолютное, но есть и отличия:

1. Фиксированный элемент тоже выпадает из потока.
2. Фиксированный элемент привязывается к определенной точке в окне браузера и остается на ней всегда, даже при прокрутке страницы.
3. Фиксированный элемент можно позиционировать с помощью свойств top, left, right, bottom, но точка отсчета всегда привязана к окну браузера.

Можете представить себе фиксированный элемент, как стикер, который наклеен на монитор. Такие элементы часто применяют для создания навигационных панелей, привязанных к верхней или нижней части окна браузера.

## z-index или кто кого перекроет [19/20]

Если в одном месте страницы оказываются несколько «абсолютных» блоков, то они перекрывают друг друга. По умолчанию выше оказывается тот блок, который расположен дальше в коде страницы.

C помощью CSS-свойства z-index можно управлять тем, как перекрываются блоки. Значением этого свойства может быть целое число. Чем больше z-index, тем выше располагается блок.

Cвойство z-index работает для элементов, у которых position задано как absolute, fixed и relative. Таким образом, «относительный» элемент может перекрывать «абсолютный».

# Мастерская: Создаём меню

## Создаём вертикальное меню [1/28]

В этом курсе мы будем пошагово создавать различные меню. От самых простых вертикальных и горизонтальных до более сложных, с выпадающими подменю.

При верстке меню первым делом размечают HTML-код. Меню — это список ссылок на страницы, поэтому и размечают меню с помощью списков.

В этой части курса мы делаем типовое меню типового корпоративного сайта. Для начала создадим список.

**Что повторить:**  
[Неупорядоченный список](https://htmlacademy.ru/courses/38/run/3)

## Добавляем ссылки [2/28]

Теперь необходимо добавить в список ссылки с адресами страниц.

Для удобства мы будем использовать ссылки с якорями, так как у нас нет страниц, на которые можно перейти. Также при щелчке на ссылку с якорем не происходит перезагрузка страницы.

Но если щелкнете и страница в мини-браузере перезагрузится, то используйте кнопку обновления мини-браузера.

**Что повторить:**  
[Что такое ссылка?](https://htmlacademy.ru/courses/40/run/1)  
[Задаем адрес ссылки](https://htmlacademy.ru/courses/40/run/2)  
[Ссылка с якорем](https://htmlacademy.ru/courses/40/run/7)

## Сбрасываем стили списка [3/28]

Разметка готова. Теперь можно переходить к оформлению меню.

Сначала добавим контейнеру меню свой CSS-класс. В качестве контейнера будем использовать список.

Каждый браузер по умолчанию отображает списки с внутреними и внешними отступами, а также добавляет маркеры к пунктам списка.

Нам нужно обнулить все отступы у списка, а также убрать маркеры.

Маркерами списков управляет свойство list-style. Маркеры исчезнут, если ему задать значение none.

**Что повторить:**

1. [Селекторы по классам](https://htmlacademy.ru/courses/42/run/3)
2. [Внутренние отступы, свойство padding](https://htmlacademy.ru/courses/44/run/4)
3. [Внешние отступы, свойство margin](https://htmlacademy.ru/courses/44/run/5)

## Оформляем контейнер меню [4/28]

После того, как стили списка сброшены, можно приступать к оформлению меню с чистого листа.

Для начала оформим контейнер меню. Обычно контейнер меню выделяют рамками или фоном. Чтобы пункты меню не прилипали к границам контейнера, добавляют внутренние отступы. В зависимости от дизайна сайта также добавляют внешние отступы и задают ширину.

Мы используем все перечисленные свойства.

**Что повторить:**

1. [Рамки](https://htmlacademy.ru/courses/44/run/6)
2. [Ширина и высота](https://htmlacademy.ru/courses/44/run/3)
3. [Cвойство background-color](https://htmlacademy.ru/courses/53/run/1)

## Оформляем пункты, простой вариант [5/28]

Простейший способ оформить пункты меню — добавить между ними отступы и задать стили ссылок.

Хорошим тоном считается задавать стили ссылок для всех состояний: по умолчанию, посещенная, при наведении и щелчке. Но для этого дизайнер должен продумать и отрисовать все состояния ссылок.

Мы пойдем простейшим путем и зададим одинаковый цвет ссылок для всех состояний (всё-таки мы только тренируемся).

Обратите внимание на способ написания селекторов. Мы используем контекстные селекторы. Перед каждым CSS-правилом, относящимся к меню, мы пишем класс меню. Такой способ называется созданием пространства имен. Он позволяет эффективно отделять друг от друга стили, относящиеся к разным блокам.

**Что повторить:**

1. [Псевдоклассы :link, :visited и :active](https://htmlacademy.ru/courses/42/run/14)
2. [Контекстные селекторы](https://htmlacademy.ru/courses/42/run/5)

## Более сложное оформление пунктов [6/28]

Простое, но аккуратное вертикальное меню готово. Теперь немного усложним его:

1. добавим в меню разделители;
2. увеличим площадь ссылок, чтобы по ним было легче попадать;
3. оформим различные состояния пунктов меню.

Большая часть стилей оформления уйдет в пункты меню, поэтому на первом шаге снова уберем отступы у контейнера и добавим их ссылкам. Ссылки сделаем блочными элементами.

**Что повторить:**

1. [Управление типом элемента, свойство display](https://htmlacademy.ru/courses/44/run/17)
2. [Стандартная блочная модель](https://htmlacademy.ru/courses/44/run/7)

## Добавляем разделители [7/28]

Теперь нужно убрать рамки и фон у контейнера. Можно просто удалить эти свойства, либо задать им значение none.

Затем добавим рамки ссылкам. А чтобы не возникало двойных рамок на стыке ссылок, добавим им отрицательный внешний отступ снизу, такой же ширины, что и рамки.

И последний штрих. Уберем подчеркивание у ссылок, так как появились разделители и от обилия горизонтальных линий рябит в глазах.

**Что повторить:**

1. [Свойство text-decoration: подчеркивание и другие эффекты](https://htmlacademy.ru/courses/43/run/7)

## Оформляем состояния пунктов [8/28]

Пункт меню может находиться как минимум в трёх состояниях:

1. спокойное состояние;
2. наведён курсор мыши;
3. активное состояние — открыта страница, которой соответствует этот пункт.

Хорошим тоном считается задавать стили для этих состояний.

С помощью псевдокласса :hover задают стили при наведении курсора мыши. Для активного состояния создают дополнительный класс, который добавляют активному пункту.

Добавим стили для состояний. Будем считать, что мы находимся на странице Услуги.

**Внимание!** Не путайте активное состояние пункта меню и псевдокласс :active у ссылок. Этот псевдокласс срабатывает в момент, когда клавиша мыши нажата на ссылке. А активное состояние пункта меню — это часть логики интерфейса.

**Что повторить:**

1. [Псевдокласс :hover](https://htmlacademy.ru/courses/42/run/12)

## Многоуровневое вертикальное меню [10/28]

Теперь задачка посложнее — создать многоуровневое меню. Такие меню часто встречаются в каталогах товаров с большим деревом категорий.

Так как уровней в меню может быть очень много, то второй, третий и более глубокие уровни оформляют максимально просто и единообразно. И только первый уровень оформляется более сложно.

В HTML многоуровневые меню размечают как обычные многоуровневые списки. И мы начнем работу с того, что добавим в исходный список подменю.

**Что повторить:**

1. [Многоуровневый список](https://htmlacademy.ru/courses/38/run/5)

## Устраняем проблемы оформления [11/28]

Подменю добавлено, но перед тем как продолжить оформление, нужно устранить возникшие проблемы.

Во-первых, все ссылки подменю подсвечены синим, так как лежат внутри элемента списка с классом active, а селектор .main-menu .active a действует на все ссылки.

Чтобы избавиться от этого эффекта, нужно изменить селектор, используя дочерние селекторы, на такой: .main-menu > .active > a. Подсвечиваться будет только ссылка верхнего уровня.

Во-вторых, нужно сбросить отступы и убрать маркеры у подменю, как вы делали в [3 задании](https://htmlacademy.ru/courses/50/run/3).

**Что повторить:**

1. [Дочерние селекторы](https://htmlacademy.ru/courses/42/run/8)

## Переносим рамки, задаём отступы подменю [12/28]

Мы оставим рамки только для пунктов меню верхнего уровня.

Чтобы сделать это, нужно убрать рамки у ссылок и задать их для элементов списка. Также отрицательный отступ снизу уберем у ссылок и добавим элементам списка.

Используем для этого дочерний селектор .main-menu > li.

Также на этом этапе мы добавим отступы для контейнера подменю, т.е. для тега ul.

## Оформляем пункты подменю [13/28]

Теперь оформим пункты подменю. Оформление будет отличаться от пунктов верхнего уровня и будет максимально простым.

С помощью какого селектора задавать стили для пунктов подменю?

Для этого можно использовать сложный контекстный селектор. Например, такой:

.main-menu li li a

Он выберет только те ссылки, которые лежат внутри двух и более элементов списка, то есть ссылки на втором уровне и ниже.

Также мы изменим поведение при наведении для подпунктов. Селектор будет очень похож на предыдущий:

.main-menu li li a:hover

## Тестируем на большой вложенности [14/28]

Оформление завершено. Теперь проверим, как ведут себя наши стили, когда меню содержит больше двух уровней вложенности.

Мы разметили трехуровневый список, но не добавили ему класс меню, поэтому он выглядит не очень красиво. Добавьте списку класс main-menu и посмотрите, как он преобразится.

## Горизонтальное меню [16/28]

Теперь из того же самого списка создадим горизонтальное меню.

Начнем с этапа, когда HTML-код уже размечен и заданы стили для контейнера. Теперь нужно преобразовать вертикальный список в горизонтальное меню.

Простейший способ сделать это — задать элементам списка строчный тип. Тогда они автоматически расположатся в одну строку. Также добавим им небольшой отступ справа.

Кстати, мы не задали ширину контейнеру, т.к. обычно горизонтальные меню занимают всю ширину макета.

**Что повторить:**

1. [Управление типом элемента, свойство display](https://htmlacademy.ru/courses/44/run/17)

## Что делать, если пункты не влезают? [17/28]

У горизонтальных меню есть одна неприятная особенность. Иногда все пункты меню не влезают в одну строку. Как решить эту проблему?

На самом деле никак. Всегда может возникнуть ситуация, когда что-то не влезло, её не избежать. Но предусматривать такую ситуацию надо.

Главное правило — ссылки в меню должны быть доступны и видимы при любых обстоятельствах. Пусть они переносятся на другую строку и меню выглядит не так красиво, но главное, что ссылки видны.

Мы можем только уменьшить вред от этого эффекта. Например, сделать так, чтобы пункты меню переносились на новую строку целиком, запретив переносы текста внутри них.

**Что повторить:**

1. [Управляем пробелами: white-space](https://htmlacademy.ru/courses/43/run/10)

## Усложненное оформление пунктов [18/28]

Теперь давайте усложним оформление. Сделаем пункты более объемными и добавим разделители по аналогии с вертикальным меню.

Для начала избавимся от фона и отступов у списка. А рамку у списка оставим, т.к. в горизонтальном меню пункты не всегда занимают всю ширину или влезают на одну строку.

Чтобы расположить пункты в одну строку и иметь возможность задавать им отступы и размеры, зададим блочно-строчный тип элементам списка.

А ссылкам внутри элементов списка зададим блочный тип и добавим отступы.

**Что повторить:**

1. [display: inline-block](https://htmlacademy.ru/courses/44/run/18)

## Завершаем оформление пунктов [19/28]

На этом шаге мы добавим рамки, избавимся от лишних отступов и оформим состояния меню.

Так как отступы мы задали ссылкам, то и рамки добавим им же. Причем нам нужно задать только рамки слева, а чтобы не было двойной рамки у самого первого пункта, добавим отрицательный внешний отступ слева.

Обнулим отступ после элементов списка, но этого недостаточно, так как пробелы после блочно-строчных элементов добавляют небольшой отступ. Чтобы от него избавиться, зададим отрицательный внешний отступ справа элементам списка.

Затем оформим состояние ссылки при наведении (добавим фон). При желании можете поэкспериментировать с оформлением активного пункта меню.

Чтобы пробелы после блочно-строчных элементов были видны, мы добавили для ссылок серый фон.

## Вертикальное меню с выпадающим подменю [21/28]

Еще один распространенный тип меню — это меню с выпадающими подменю. Обычно в таких меню только два уровня, но встречаются и многоуровневые, когда из выпадающего подменю может выпадать еще одно и так далее.

Разметка таких меню простая — обычный многоуровневый список, который вы уже делали в этом курсе. При создании такого меню необходимо использовать абсолютное и относительное позиционирование.

Обычно элементам списка li верхнего уровня задают относительное позиционирование, а для выпадающего меню — абсолютное.

Начнем создавать такое меню с этапа, когда готова разметка и базовые стили списков и пунктов. Для удобства фон подменю сделан светло-коричневым, а подменю есть только у одного пункта.

**Что повторить:**

1. [Относительное позиционирование](https://htmlacademy.ru/courses/45/run/2)
2. [Абсолютное позиционирование](https://htmlacademy.ru/courses/45/run/8)

## Позиционируем выпадающее подменю [22/28]

Затем выпадающее подменю располагают в том месте, куда оно будет «выпадать». Лучше располагать так, чтобы оно немного «нависало» над родительским пунктом.

Также нужно установить для подменю z-index больше, чем у родительских пунктов, чтобы подменю располагалось выше и перекрывало их при появлении.

Так как абсолютное позиционирование сбрасывает ширину, то можно установить ширину для подменю.

**Что повторить:**

1. [Точка отсчета координат](https://htmlacademy.ru/courses/45/run/14)
2. [Тренируемся задавать координаты](https://htmlacademy.ru/courses/45/run/15)
3. [z-index или кто кого перекроет](https://htmlacademy.ru/courses/45/run/19)

## Отображаем подменю при наведении [23/28]

Последний этап — добавление интерактивности.

Сначала прячем выпадающее подменю по умолчанию, с помощью display: none;.

Затем делаем так, чтобы при наведении мыши на пункт меню, внутри которого расположено подменю, оно появлялось. Для этого используем контекстный селектор в сочетании с псевдоклассом :hover и display: block;.

**Что повторить:**

1. [display: none](https://htmlacademy.ru/courses/44/run/22)
2. [Динамические эффекты с помощью :hover](https://htmlacademy.ru/courses/42/run/13)

## Горизонтальное меню с выпадающим подменю [24/28]

Теперь построим горизонтальное меню, у которого будут вертикальные выпадающие подменю.

Разметка стандартная — двухуровневый список.

Алгоритм оформления почти такой же, как и в предыдущем случае. Сначала задаем режимы позиционирования, затем стили для подменю, затем механизм выпадания.

Так как мы хотим, чтобы выпадающее меню было вертикальным, то надо изменить селектор .main-menu li на .main-menu > li, чтобы блочно-строчными были только элементы списка верхнего уровня.

## Оформляем выпадающее меню [25/28]

Оформим выпадающее меню.

Во-первых, зададим ширину для подменю. Во-вторых, зададим координаты. Сейчас подменю уже располагается хорошо, поэтому верхнюю координату трогать не будем, а зададим только координату слева.

Пункты подменю тоже оформим очень просто. Во-первых, зададим для ссылок рамку со всех сторон, а не только слева. И, во-вторых, добавим отрицательный отступ снизу, чтобы не было двойных рамок.

## Включаем механизм выпадания [26/28]

По аналогии с [23 заданием](https://htmlacademy.ru/courses/50/run/23) делаем так, чтобы подменю появлялось при наведении мыши на родительский пункт меню.

Прячем подменю по умолчанию и отображаем с помощью контекстного селектора с псевдоклассом :hover.

## Состояние «открытое подменю» [27/28]

Иногда нужно сделать так, чтобы выпадающее подменю было открыто по умолчанию (а не только при наведении мыши). Как этого добиться?

Сначала к элементу списка в меню верхнего уровня, где должно быть открыто подменю, добавим класс opened.

Затем используем селектор: .main-menu .opened .sub-menu. И в нём пропишем свойство, которое отображает подменю.

Очень похоже на выделение активного пункта меню. Кстати, чтобы выделить сам пункт меню с открытым подменю, можно использовать вот такой селектор .main-menu > .opened > a, т.е. «ссылка, которая лежит непосредственно в элементе с классом opened».

# [Мастерская: декоративные элементы](https://htmlacademy.ru/courses/55)

## Социальные кнопки, шаг 1 [1/18]

Начнём прохождение курса с простой задачи — вёрстки кнопок социальных сетей.

Обычно такие кнопки являются просто ссылками на страницы компании в социальных сетях. Стилизация работающих «лайков» является более сложной задачей.

Начнём с разметки. Нужно будет добавить в код три ссылки и задать каждой из них два класса. Один класс будет общим и будет отвечать за форму и размер кнопок. Второй класс будет уникальным и будет задавать фон и изображение соответствующей соцсети.

**Что повторить:**

1. [Что такое ссылка?](https://htmlacademy.ru/courses/40/run/1)
2. [Каскадность. Коктейль из классов](https://htmlacademy.ru/courses/41/run/12)

## Социальные кнопки, шаг 2 [2/18]

Мы будем выстраивать кнопки в ряд и задавать им размеры, поэтому нужно использовать блочно-строчный тип элемента.

Затем зададим фон и посмотрим, как ведут себя кнопки и подпись. Для того, чтобы красиво выровнять подпись «Соцсети» и кнопки, зададим вертикальное выравнивание по середине.

На этом этапе мы задаем общие стили кнопок, поэтому записываем их в CSS-правило для класса social.

**Что повторить:**

1. [Блочно-строчный тип](https://htmlacademy.ru/courses/44/run/18)
2. [Ширина и высота](https://htmlacademy.ru/courses/44/run/3)
3. [Внешние отступы](https://htmlacademy.ru/courses/44/run/5)
4. [Цвет фона](https://htmlacademy.ru/courses/53/run/1)
5. [Вертикальное выравнивание](https://htmlacademy.ru/courses/43/run/12)

## Социальные кнопки, шаг 3 [3/18]

У нас есть 3 серых ссылки-квадрата, осталось покрасить их в цвета соцсетей и добавить иконки.

В этом курсе мы часто будем использовать спрайты. Вот изображение-спрайт с иконками соцсетей:

Спрайт будет общим для всех ссылок, поэтому фоновое изображение зададим для класса social. А для уникальных классов соцсетей переопределим расположение фонового изображения и цвет фона.

Обратите внимание, что CSS-правила для конкретных соцсетей расположены в коде ниже, чем правило, задающее общий вид. Это нужно, чтобы работало переопределение CSS-свойств.

**Что повторить:**

1. [Фоновое изображение](https://htmlacademy.ru/courses/53/run/2)
2. [Повторение фона](https://htmlacademy.ru/courses/53/run/3)
3. [Расположение фона](https://htmlacademy.ru/courses/53/run/4)
4. [Спрайты](https://htmlacademy.ru/courses/53/run/15)
5. [Каскадность и приоритеты](https://htmlacademy.ru/courses/41/run/11)

## Переключатель страниц, шаг 1 [4/18]

Теперь потренируемся верстать переключатель страниц. Обычно разметка переключателей очень проста: это один див, внутри которого расположены ссылки с номерами страниц и ссылки «Вперёд/Назад».

Иногда внутри переключателя используют спаны, например, для «многоточий». Также переключатели иногда верстают списками.

Для начала зададим для ссылок и спанов блочно-строчный тип, размеры, внутренние отступы и фон.

Обратите внимание, что мы задаём не ширину, а минимальную ширину min-width, т.к. номера страниц могут быть длинными: 1 и 1000. Благодаря сочетанию паддингов и минимальной ширины, номер страницы будет всегда смотреться хорошо.

**Что повторить:**

1. [Внутренние отступы](https://htmlacademy.ru/courses/44/run/4)

## Переключатель страниц, шаг 2 [5/18]

Теперь оформим текст переключателей.

Сначала зададим горизонтальное выравнивание по центру. Затем зададим вертикальное выравнивание по середине. Чтобы был виден эффект вертикального выравнивания, нужно задать высоту строки такую же, как высота элемента, т.е. 40px.

Затем зададим цвет ссылок, уберем подчеркивание и сделаем шрифт полужирным.

**Что повторить:**

1. [Горизонтальное выравнивание](https://htmlacademy.ru/courses/43/run/11)
2. [Вертикальное выравнивание](https://htmlacademy.ru/courses/43/run/12)
3. [Высота строки](https://htmlacademy.ru/courses/43/run/14)
4. [Текстовые эффекты](https://htmlacademy.ru/courses/43/run/7)
5. [Выделение жирным](https://htmlacademy.ru/courses/43/run/3)

## Переключатель страниц, шаг 3 [6/18]

Займемся переключателями «вперед» и «назад». Пока что мы используем текстовые надписи. Но мы их спрячем и заменим на стрелочки.

Первым делом добавим к переключателям классы prev и next. Чтобы спрятать текст, используем свойство text-indent с большим отрицательным значением.

Затем отделим переключатели от номеров страниц с помощью отступов и с помощью уже знакомой техники спрайтов добавим на фон стрелочки. Вот спрайт на тёмном фоне:

## Переключатель страниц, шаг 4 [7/18]

Переключатель страниц почти готов. Но хороший верстальщик и хороший дизайнер всегда думают о мелочах.

В этом задании мы поработаем над состояниями элементов переключателя.

Первая очевидная деталь относится к номерам страниц. Надо как-то выделять текущую страницу. Для этого будем использовать класс current, для которого зададим зелёный фон.

Менее очевидная деталь относится к состояниям кнопок-стрелочек «вперед» и «назад». Когда мы находимся на первой странице, кнопка «назад» должна быть неактивна, а когда на последней странице — неактивна кнопка «вперед».

Оформим неактивные кнопки с помощью класса disabled, который будет делать кнопку полупрозрачной.

1. **Цель 1**Добавьте ссылке с классом prev класс disabled.
2. **Цель 2**Добавьте ссылке на первую страницу класс current.
3. **Цель 3**Для .current задайте цвет фона #16a085.
4. **Цель 4**Для .disabled задайте 30% непрозрачность opacity: 0.3;.
5. **Цель 5**Задайте ссылкам цвет фона #2ecc71 при наведении.

## Переключатель страниц, тестирование [8/18]

Теперь оттестируем наш переключатель в разных состояниях.

У нас есть три переключателя, у которых разные текущие страницы. Ваша задача заключается в том, чтобы правильно задать элементам переключателей те классы, которые мы добавляли в прошлом задании.

## Контакты, шаг 1 [10/18]

В этой серии заданий мы оформим невзрачный и скучный блок контактов.

Блоки с контактной информацией, похожие на тот, который вы видите в мини-браузере, встречаются на многих сайтах. Штука нехитрая — пишем адрес, сайт и email, а потом добавляем переносы строк с помощью <br>.

Как улучшить оформление этого блока?

Во-первых, выделить заголовок и добавить отступы между элементами контактов. Во-вторых, добавить для каждого элемента небольшие иконки.

Давайте начнем.

## Контакты, шаг 2 [11/18]

Заголовок и отступы сделали блок контактов более структурированным.

Отступы слева, как вы уже догадались, предназначены для размещения иконок.

Иконки будем размещать по одной из классических техник: «абсолютные элементы + спрайты».

В каждом из блоков .item есть пустой див .icon, в котором и будет отображаться иконка.

Специальные дивы в этом задании мы будем использовать для наглядности. Лучше иконки выносить в фон псевдоэлементов ::before и ::after, чтобы не «засорять» разметку.

Итак, давайте зададим этим дивам абсолютное позиционирование, фиксированные размеры и расположим в верхнем левом углу родителя (для этого мы задавали относительное позиционирование для .item).

Осталось задать для .icon картинку-спрайт в качестве фона. Вот она:

## Контакты, шаг 3 [12/18]

На последнем шаге нужно задать каждому элементу контактов свою иконку.

Сделаем это так же, как в задании про социальные кнопки. Сначала добавим каждому из блоков .icon свой уникальный класс. А затем в CSS переопределим положение фона для этих классов.

И финальный штрих — зададим цвет ссылок, который сочетается с цветом иконок.

## Комментарии, шаг 1 [13/18]

В последней серии заданий мы сверстаем список комментариев.

Кстати, обратите внимание на CSS. В предыдущих заданиях мы создавали пространства имён классов в CSS с помощью контекстных селекторов. Например:

.contacts .title {}

.contacts .item {}

О пространствах имен уже [говорилось](https://htmlacademy.ru/courses/50/run/5) в курсе про меню.

В этой серии заданий мы будем использовать другой способ создания пространства имён. Он заключается в том, что мы будем добавлять к названию каждого класса префикс, например:

.comment-avatar {}

.contacts-title {}

Оба способа хороши и используются в зависимости от предпочтений. Самое главное — использовать пространства имён.

В этом задании мы начнём оформлять блоки аватара и автора комментария.

## Комментарии, шаг 2 [14/18]

Продолжаем работу. На этом шаге оформим сам текст комментария и дату комментария.

Обычно менее важную информацию стараются приглушить. Для этого используют следующие приёмы: уменьшают размер шрифта, задают менее контрастный цвет текста или сочетают эти эффекты. Простейшие CSS-свойства помогают эффективно расставлять акценты в дизайне.

В нашем случае дата комментария не столь важна, поэтому её и приглушим.

## Комментарии, шаг 3 [15/18]

На этом шаге оформляем кнопку ответа на комментарий.

Все приёмы вам уже знакомы: кнопка будет абсолютно спозиционированным элементом с фиксированными размерами. На фон ей зададим иконку. Затем разместим кнопку в правом верхнем углу комментария.

Блок комментария обладает относительным позиционированием, поэтому проблем с размещением иконки не возникнет.

## Комментарии, шаг 4 [16/18]

Сейчас подготовимся к позиционированию аватара.

Аватар будет располагаться в левом верхнем углу и частично перекрывать комментарий. Чтобы аватар не перекрыл имя автора и текст, мы увеличим паддинги слева у соответствующих элементов.

Мы увеличиваем отступ справа у автора, т.к. в правом верхнем углу комментария расположена кнопка ответа (спозиционированная абсолютно). Чтобы длинное имя автора не залезло под кнопку, мы и добавляем паддинг, размер которого чуть больше, чем кнопка.

## Комментарии, шаг 5 [17/18]

Теперь осталось только задать аватару абсолютное позиционирование и разместить его сверху и слева.

Достаточно необычные комментарии, но способ их создания очень простой.

# [Знакомство с LESS](https://htmlacademy.ru/courses/85)

## Введение [1/14]

CSS-препроцессоры — это «программистский» подход к CSS. Они позволяют при написании стилей использовать свойственные языкам программирования приёмы и конструкции: переменные, вложенность, наследуемость, циклы, функции и математические операции. Синтаксис препроцессоров похож на обычный CSS. Код, написанный на языке препроцессора, не используется прямо в браузере, а преобразуется в чистый CSS-код с помощью специальных библиотек.

Три самых известных препроцессора — это [LESS](http://lesscss.org/), [SASS](http://sass-lang.com/) и [Stylus](http://learnboost.github.io/stylus/). Они во многом похожи между собой, но имеют и ключевые различия. В этом и последующих курсах цикла мы рассмотрим препроцессор LESS.

В курсах по LESS мы будем шаг за шагом строить небольшой  фреймворк компонентов. Когда он будет готов, мы сможем собирать из компонентов цельные интерфейсы. Компонентный подход позволяет структурировать большие объемы кода и легко масштабировать проекты. Препроцессор в этом деле — хорошее подспорье.

В этой серии курсов редактор CSS заменён на редактор LESS. Чтобы увидеть скомпилированный из LESS кода CSS код, можете использовать кнопку **CSS**.

Итак, теперь вы знаете всё, чтобы начать!

## Переменные, шаг 1 [2/14]

Цветовая схема — основа любого дизайна в вебе. Применяя цветовое кодирование, можно сделать интерфейс более понятным. Поэтому первым делом при создании нашего мини-фреймворка давайте займёмся цветовой схемой основных элементов интерфейса. В этой задаче LESS нам очень поможет.

В прошлом задании в коде встречалась подобная запись:

@navy: #1d365d;

Так в LESS описываются переменные. Синтаксис переменных такой:

@название\_переменной: значение\_переменной;

Создав переменную один раз, можно использовать её в любом месте кода. Например:

background-color: @navy;

color: @navy;

border-color: @navy;

Во всех местах, где указана переменная, LESS заменит строку @navy на #1d365d. Теперь, если понадобится изменить цвет, не нужно искать все его объявления в файле, а достаточно просто изменить значение переменной в одном месте.

Попробуем использовать переменные в коде: зададим с помощью переменной основной цвет нашей цветовой схемы.

## Переменные, шаг 2 [3/14]

Переменные можно объявлять как «снаружи» правил, так и «внутри». В случае «внутреннего» объявления переменная будет доступна только внутри правила, в котором она объявлена:

.rule-1 {

@align: right;

text-align: @align; // text-align задаётся значение right

}

.rule-2 {

text-align: @align; // в этом месте произойдёт ошибка

}

Если переменная объявлена и «внутри» правила, и «снаружи» — LESS применит «внутреннее» значение.

@align: left;

.rule-1 {

@align: right;

text-align: @align; // text-align задаётся значение right

}

.rule-2 {

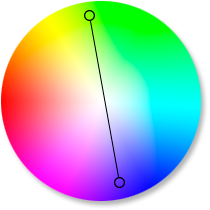
text-align: @align; // text-align задаётся значение left

}

Таким образом можно «переопределять» глобальные переменные в локальном контексте.

Проверим это на практике.

## Цветовые функции, шаг 1 [4/14]



Итак, мы задали базовый цвет для нашей схемы, от него мы будем отталкиваться при выборе других цветов. Давайте узнаем, как это сделать.

Все цвета модели RGB можно расположить на цветовом колесе, где они плавно переходят друг в друга.

С помощью LESS-функции spin можно повернуть цветовое колесо на определённый угол относительно заданного цвета и получить новый цвет. Функция принимает два параметра, синтаксис её такой:

spin(цвет, угол\_поворота)

Цвет можно задавать в любом цветовом формате. Значение угла может быть как положительным, так и отрицательным. При положительном угле функция повернёт колесо по часовой стрелке, при отрицательном — против. Примеры:

color: spin(red, 90); // цвет повернётся от красного на 90° по часовой

border-color: spin(#f0f, -45); // цвет на 45° от #f0f против часовой

Противоположный цвет на колесе называется *комплементарным*. Он находится под углом 180°к заданному цвету. Комплементарные цвета используют для создания контраста.

Наш основной цвет находится в сине-фиолетовой области цветового круга. Давайте для него вычислим комплементарный цвет с помощью функции spin.

## Вложенные правила, шаг 1 [5/14]

Отвлечёмся ненадолго от цвета и рассмотрим ещё одну замечательную особенность LESS — вложенные правила. Они позволяют избавиться от дублирования одинаковых названий в коде и делают его более структурированным. Например, вот такой код:

.super-class-name {

color: #ffffff;

}

.super-class-name a {

text-decoration: none;

}

.super-class-name a span {

font-size: 1em;

}

можно более кратко и без повторов написать, используя вложенность:

.super-class-name {

color: #ffffff;

a {

text-decoration: none;

span {

font-size: 1em;

}

}

}

То есть вложенные правила просто пишутся внутри других правил. Из цепочек вложенных правил LESS сам составляет итоговые селекторы.

Попробуем написать вложенное правило.

## Вложенные правила, шаг 2 [6/14]

С помощью вложенных правил можно не только обращаться к дочерним элементам, но и составлять по частям комплексные названия классов. Например, следующий код:

.super-button-red {

color: red;

}

.super-button-blue {

color: blue;

}

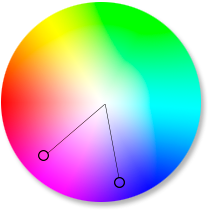
можно записать проще c помощью вложенных правил:

.super-button {

&-red { color: red; }

&-blue { color: blue; }

}

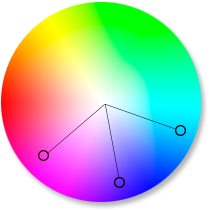


То есть, если перед вложенным правилом поставить амперсанд &, то оно станет *родственным* родительскому, и LESS подставит родительский селектор вместо амперсанда.

Воспользуемся этой особенностью для структурирования кода нашей цветовой схемы. Cоздадим второй цвет палитры — яркий акцентный цвет ошибки, повернув цветовое колесо на 60° по часовой стрелке относительно базового, чтобы цвет сместился в красную область.

Сохраним полученный цвет в переменной и зададим с её помощью фон второго контейнера.

## Математические операции, шаг 1 [7/14]



Теперь давайте создадим третий цвет. Его «сдвинем» по кругу на 60° в другую сторону от основного. Он светлее и будет использоваться для выделения второстепенной информации и элементов.

Чтобы «сдвинуть» цвет против часовой стрелки, нужно в функцию spin передать отрицательное значение переменной @distance.

Над любыми численными значениями в LESS-коде можно произвести математические операции сложения, вычитания, умножения или деления:

padding-top: 10px + 20; // = 30px

padding-bottom: 100px - 50; // = 50px

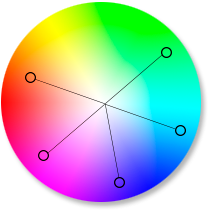
font-size: 2em \* 2; // = 4em

left: 50% / 2; // 25%

LESS выполнит математическую операцию и вернёт в CSS уже вычисленное значение. Единицы измерения всегда берутся от первого параметра в выражении.

Чтобы вычислить отрицательное значение @distance просто умножим переменную на -1.

## Математические операции, шаг 2 [8/14]



Осталось добавить ещё два цвета. Чтобы их вычислить, мы «повернём» круг на двойное расстояние @distanceот базового цвета в обе стороны.

Получившиеся цвета будут комплементарными по отношению к инфо-цвету и цвету ошибки. Это именно то, что нужно: к примеру, по смыслу цвет ошибки полностью противоположен цвету успеха, поэтому они должны быть максимально контрастны.

Из математических операций используем умножение дистанции на 2.

Давайте же завершим нашу палитру.

## Математические операции, шаг 3 [9/14]

И теперь, когда палитра цветов построена, можно немного «поиграть»  с базовым цветом и посмотреть, как вместе с ним будут меняться остальные.

Для этого давайте просто поменяем параметры в RGB-записи цвета в переменной @base-colorматематическими операциями. Кстати, к значению цвета «целиком» тоже можно применять операции. В случае суммы числа и RGB-записи цвета слагаемое будет прибавляться к каждому цветовому каналу одновременно:

rgb(10, 10, 10) + 10

// то же самое, что

rgb(20, 20, 20)

## Цветовые функции, шаг 2 [10/14]

Продолжим создавать библиотеку компонентов: с помощью цветов из разработанной палитры зададим состояния кнопок. И по ходу дела разберём другие функции LESS для работы с цветом.

Сначала зададим базовый цвет для фона кнопок и добавим его вариации по наведению и нажатию на кнопку. По наведению будем делать кнопку чуть светлее, а в момент нажатия — темнее. Для этого воспользуемся LESS-функциями lighten и darken. Их синтаксис одинаковый:

color: lighten(red, 50%); // светлее red на 50%

color: darken(blue, 25%); // темнее blue на 25%

Второе значение задаётся в процентах от 0% до 100%. При задании 100% в lighten функция возвращает полностью белый цвет, а 100% для darken — полностью чёрный. То есть эти функции «смешивают» заданный цвет с белым или чёрным.

Итак, давайте зададим кнопкам нужные цвета и снова воспользуемся вложенными правилами. Чтобы задать смену цвета и для демонстрационного класса btn-hover, и для псевдокласса btn:hover, нужно прописать вложенные правила в LESS так:

.btn {

&-hover,

&:hover {

color: red;

}

}

Этот LESS преобразуется в следующий CSS:

.btn-hover,

.btn:hover {

color: red;

}

## Цветовые функции, шаг 3 [11/14]

Ещё две LESS-функции для работы с цветом: saturate и desaturate. Они увеличивают и уменьшают насыщенность заданного цвета. Их синтаксис такой же как и у lighten/darken:

color: saturate(green, 20%); // green насыщеннее на 20%

color: desaturate(blue, 50%); // blue бледнее на 50%

Функцию desaturate мы используем для создания стиля «отключенной» кнопки. Совместно с desaturate воспользуемся lighten, чтобы сделать кнопку бледной и светлой. Одновременно функции цвета можно использовать так:

// цвет светлее красного на 50% и насыщеннее на 20%

color: saturate(lighten(red, 50%), 20%);

//цвет темнее синего на 20% и бледнее на 50%

color: desaturate(darken(blue, 20%), 50%);

То есть функции можно «вкладывать» друг в друга, используя их в качестве аргументов.

## Цветовые функции, шаг 4 [12/14]

Теперь перейдём к созданию цветных информационных сообщений. Они будут кодироваться теми же цветами из палитры: основным, контрастным, информационным, предупреждающим и успешным. С помощью цветовых функций и их комбинаций давайте подберём подходящие оттенки для элементов сообщений.

Фоновый цвет блоков сделаем светлее основного. Цвет границ — немного темнее и контрастнее фонового цвета. Цвет текста — ещё более темным, чем цвет фона.

Воспользуемся уже известными нам функциями lighten, darken, spin и их комбинациями.

## Цветовые функции, шаг 5 [13/14]

Теперь создадим стили для остальных блоков сообщений.

Будем использовать те же функции, но только с другими цветами, от которых ведётся отсчёт. В этом задании переменные уже созданы, но они привязаны к базовому цвету. Нужно будет лишь заменить в функциях переменную базового цвета на переменные подходящих цветов.

Вынос всех цветов в переменные — очень прагматичный подход. Переменные можно легко собрать в одном файле, что позволяет гибко конфигурировать внешний вид базовых компонентов.

# [Знакомство с SVG](https://htmlacademy.ru/courses/130)

## Привет, SVG! [1/15]

SVG — это формат векторной графики. В отличие от растровой графики — PNG, GIF, JPEG — SVG может растягиваться и сжиматься без потери качества, то есть такие картинки будут одинаково чёткими и на обычных экранах, и на ретине.

Ещё одно из достоинств SVG — человекопонятный код: его можно не только прочитать, но и написать руками. Можно открыть файл и отредактировать его без использования графического редактора, можно самому написать простую картинку.

Также SVG-элементы можно оформить с помощью CSS и добавить им интерактивности с помощью JavaScript, а кроме того, SVG [достаточно хорошо поддерживается](http://caniuse.com/#search=svg) всеми современными браузерами, и его уже можно активно использовать.

Давайте познакомимся с ним поближе. Вот простой пример кода:

<svg>

<circle r="50" cx="50%" cy="50%" fill="yellowgreen"/>

</svg>

SVG-элемент вставляется с помощью тега svg, внутри которого уже находится остальное содержимое: фигуры, картинки или текст.

Содержимое в этом примере — это кружок (circle) зелёного цвета (fill="yellowgreen"). Вот так будет выглядеть действие этого кода в браузере:

SVG можно встраивать несколькими разными способами, мы рассмотрим их позже, а сейчас будем вставлять его непосредственно в код страницы.

## Рисуем прямоугольник [2/15]

В SVG есть несколько способов нарисовать фигуру, для простых фигур есть свои теги.

Например, прямоугольник рисуется с помощью тега rect. Код простого прямоугольника выглядит так:

<rect width="150" height="100"/>

​Обратите внимание: все теги в SVG должны быть закрыты, то есть должно быть так: <rect .../> или так: <rect ...></rect>. Мы будем использовать первый способ.

Результат:

Атрибуты width и height управляют, соответственно, шириной и высотой фигуры. Значения можно задавать и в пикселях, и в процентах.

Для значений в пикселях после значения не нужно писать px, потому что пиксели — единица измерения, используемая в SVG по умолчанию. Проценты рассчитываются относительно размеров всего SVG-изображения: горизонтальные значения относительно ширины, вертикальные — относительно высоты.

В современных браузерах размерами и положением фигур нельзя управлять через CSS, но эта возможность появится в будущем.

## Координаты прямоугольника [3/15]

Чтобы задать координаты прямоугольника, используются атрибуты x и y:

<rect width="50%" height="100" x="20" y="50"/>

Координаты определяют положение верхнего левого угла фигуры.

## Скругление углов [4/15]

Скруглением углов прямоугольника управляют параметры rx и ry. Атрибут rx задаёт скругление по горизонтали, а ry — по вертикали. Если атрибут ry не задан, он будет равен rx.

Пример кода:

<rect width="50%" height="100" rx="50" ry="20"/>

В [задании курса «Рамки и фоны, часть 2»](https://htmlacademy.ru/courses/88/run/12) описан похожий эффект скругления углов, но с помощью CSS-свойства border-radius.

## Многоугольники [6/15]

В SVG можно рисовать не только четырёхугольники, но и многоугольники, это делается с помощью тега polygon. Пример кода:

<polygon points="70,5 90,41 136,48 103,80 111,126 70,105 29,126 36,80 5,48 48,41"/>

И результат:

В атрибуте points задаются координаты вершин фигуры. Каждая координата задаётся по x и y. Координаты в points нельзя задавать в процентах.

## Рисуем окружность [8/15]

Окружность рисуется с помощью тега circle. Пример кода:

<circle r="50"/>

Атрибут r — радиус окружности.

В отличие от предыдущих фигур, положение окружности в пространстве определяется координатами центра фигуры: атрибут cx задаёт положение по горизонтальной оси, cy — по вертикальной.

По умолчанию координаты центра окружности равны 0,0, поэтому она находится в верхнем левом углу. Подвинем фигуру:

<circle r="50" cx="100" cy="50%"/>

Значения можно задавать как в пикселях, так и в процентах. Процентные значения рассчитываются относительно размеров SVG-элемента.

Радиус и координаты можно задавать только атрибутами, с помощью CSS это сделать нельзя.

## Рисуем эллипс [10/15]

Эллипс рисуется почти так же, как круг, но у него два радиуса: по горизонтальной оси — rx, и по вертикальной — ry.

<ellipse rx="30" ry="40%"/>

Расположение эллипса, так же как и для circle, задаётся с помощью cx и cy.

<ellipse rx="30" ry="40%" cx="50%" cy="50%"/>

## Рисуем линии [12/15]

Линии рисуются с помощью тега line. Координаты начала линии задаются атрибутами x1 и y1, координаты конца — атрибутами x2 и y2. Координаты можно задавать в процентах.

Пример кода:

<line x1="220" y1="10" x2="20" y2="130"/>

Так как линия не образует фигуру с внутренним контуром, для отображения ей нужно задать не заливку, а обводку. Обводкой управляют два атрибута: stroke и stroke-width. Атрибут strokeзадаёт цвет обводки, stroke-width — толщину линии.

<line x1="220" y1="20" x2="20" y2="90" stroke="violet" stroke-width="5" />

Результат:

Можно задать только цвет линии, тогда толщина обводки по умолчанию будет равна одному пикселю.

## Рисуем ломаные линии [14/15]

Ломаные линии рисуются с помощью тега polyline. Координаты точек на линии задаются в атрибуте points, как для polygon.

Пример кода:

<polyline points="10,135 100,10 55,135 10,10 105,135"/>

Результат:

Разница между polygon и polyline заключается в поведении обводки: у многоугольника обводка замыкается сама по себе (левая фигура), а у ломаной линии — остаётся незамкнутой (фигура справа):

# [Оформление SVG-фигур](https://htmlacademy.ru/courses/187)

## Заливки [1/20]

В [курсе про фигуры](https://htmlacademy.ru/courses/130) вы уже немного познакомились с возможностями оформления векторных фигур, теперь мы рассмотрим этот вопрос подробнее.

SVG-фигуры имеют богатые возможности оформления: им, как и HTML-элементам, можно задавать заливку цветом, градиентом или картинкой, но помимо этого также можно управлять отдельно прозрачностью заливки и обводки, а в качестве заливки можно использовать, например, текст.

Также интересные возможности имеет обводка. Например, можно управлять видом пунктирной обводки и сделать обводку точками, пунктиром или морзянкой, а ещё обводку можно сделать не только цветом, но также градиентом или картинкой.

Начнём с заливки. Если она не задана, по умолчанию фигура заполняется чёрным цветом:

<circle r="60" cx="150" cy="50%"></circle>

Цвет заливки задаётся атрибутом fill:

<circle r="60" cx="150" cy="50%" fill="gold"></circle>

либо аналогичным свойством в CSS:

circle {

fill: gold;

}

Результат будет одинаковым:

Цвет можно задавать в любом удобном формате.

1. **Цель 1**Первой фигуре задайте заливку атрибутом в формате HEX fill="#ff7700"
2. **Цель 2**Второй — атрибутом в формате HSLA fill="hsla(50, 100%, 50%, 0.75)"
3. **Цель 3**Третьей фигуре задайте заливку именованным цветом fill: yellowgreen в редакторе CSS.

## Прозрачность заливки [2/20]

Управлять прозрачностью заливки можно с помощью свойства fill-opacity. Прозрачность также можно задавать как атрибутом, так и через CSS.

Значение задаётся числом от 0 до 1, например:

<rect width="150" height="100" fill="gold" fill-opacity="0.5"></rect>

rect {

fill: gold;

fill-opacity: 0.5;

}

Прозрачность работает для всех видов заливок, в том числе для градиентов и паттернов:

## Отсутствие заливки [3/20]

Иногда бывает нужно полностью убрать заливку, например, если вам нужен только контур фигуры. Это можно сделать ключевым словом none, результатом будет полная прозрачность фигуры.

## Обводки [5/20]

Обводки задаются с помощью нескольких атрибутов, причём цвет и толщина обводки задаются отдельно. Цвет задаётся атрибутом stroke:

<circle r="60" cx="150" cy="50%" fill="none" stroke="orange"></circle>

либо через CSS:

circle {

stroke: orange;

}

Результат будет одинаковым, у фигуры появится обводка толщиной один пиксель:

## Толщина обводки [6/20]

Для однопиксельной обводки достаточно задать только цвет в stroke. Если же нужно управлять толщиной обводки, это делается с помощью свойства stroke-width, также атрибутом или через CSS:

<circle r="60" cx="150" cy="50%" fill="none" stroke="orange"

stroke-width="5"></circle>

или:

circle {

stroke: orange;

stroke-width: 5;

}

Результат:

Короткой записи нет, поэтому цвет и толщина всегда задаются отдельно.

Если задавать значение в процентах, они будут рассчитываться не от размеров фигуры, а относительно размеров всего SVG, что может давать непредсказуемый результат.

Если обводке задана толщина, но не задан цвет, обводка не отобразится.

## Прозрачность обводки [7/20]

Прозрачность задаётся свойством stroke-opacity со значениями от 0 до 1, например: stroke-opacity="0.5".

Либо через CSS:

rect {

stroke-opacity: 0.5;

}

## Концы линий [9/20]

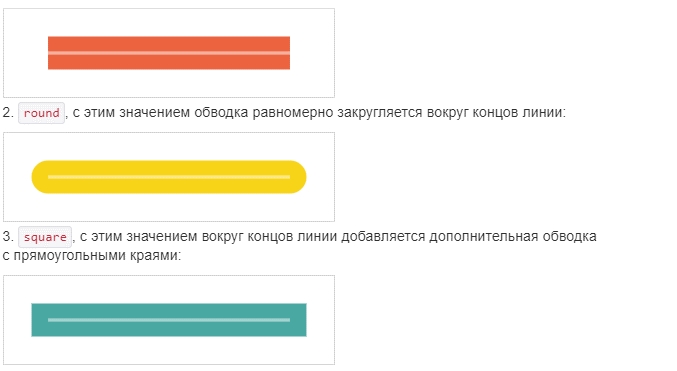
В SVG можно указывать, как ведёт себя обводка на концах линий, это делается с помощью свойства stroke-linecap. В примерах ниже мы добавили вспомогательные светлые линии с обводкой и показали возможные значения свойства stroke-linecap:

1. butt — значение по умолчанию. С этим значением обводка просто заканчивается на концах линии:

2. round, с этим значением обводка равномерно закругляется вокруг концов линии:

3. square, с этим значением вокруг концов линии добавляется дополнительная обводка с прямоугольными краями:

В CSS внешним видом обводки на концах линий можно управлять с помощью аналогичного свойства stroke-linecap.



## Вид сгибов [10/20]

Свойство stroke-linejoin позволяет управлять видом обводки на сгибах линий.

Аналогично [прошлому заданию](https://htmlacademy.ru/courses/187/run/9) мы добавили «воображаемую» светлую линию в примеры. Итак, возможные значения свойства stroke-linejoin:

1. miter — значение по умолчанию. Обводка в месте сгиба линии никак не видоизменяется.

2. round, обводка в месте сгиба линии равномерно закругляется.

3. bevel, обводка в месте сгиба линии складывается как лента.

В CSS внешним видом обводки на сгибах линий можно управлять с помощью аналогичного свойства stroke-linejoin.



## Пунктирные линии, часть 1 [12/20]

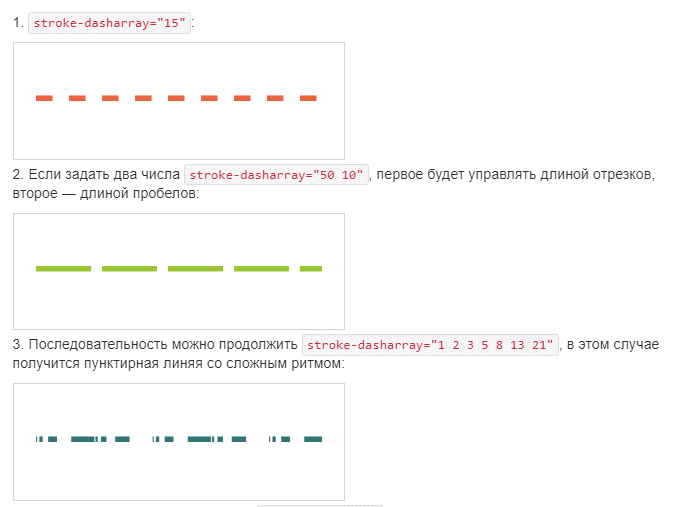
В SVG можно управлять видом пунктирных линий, это делается с помощью свойства stroke-dasharray. В качестве значения задаётся длина отрезков и пробелов между ними. Можно задать одно число, в этом случае получится пунктирная линия, состоящая из отрезков и пробелов одинаковой длины:

1. stroke-dasharray="15":

2. Если задать два числа stroke-dasharray="50 10", первое будет управлять длиной отрезков, второе — длиной пробелов:

3. Последовательность можно продолжить stroke-dasharray="1 2 3 5 8 13 21", в этом случае получится пунктирная линяя со сложным ритмом:

Аналогично работает CSS-свойство stroke-dasharray.



## Пунктирные линии, часть 2 [13/20]

Видом обводки круглой фигуры тоже можно управлять.

В отличие от прямоугольных форм сегменты пунктирной обводки круглой формы могут иметь неравномерную форму в зависимости от толщины обводки.

Давайте попробуем создать пунктирную обводку окружности и будем менять толщину, чтобы проследить, как меняется форма сегментов обводки.

1. **Цель 1**Задайте фигуре .shape-stroke ширину обводки 10 и свойство stroke-dasharray со значением 10.
2. **Цель 2**Измените значение stroke-dasharray на 20.
3. **Цель 3**Измените значение ширины обводки на 50,
4. **Цель 4**а затем на 100.



## Спасательный круг, часть 1 [15/20]

В следующей серии заданий мы нарисуем спасательный круг с помощью простых форм, которые уже изучили.

Объект будет состоять из окружностей с применёнными к ним обводками.

Тему окружностей в SVG мы рассматривали в [прошлом курсе](https://htmlacademy.ru/courses/130/run/8).

## Спасательный круг, часть 2 [16/20]

Отлично, первый «слой» объекта готов. Он будет выступать серой рамкой спасательного круга.

Давайте теперь создадим второй слой, который будет являться «раскрашенной» частью круга. Обводка этого второго круга будет немного меньше, чем первого.

## Спасательный круг, часть 3 [17/20]

Второй «слой» готов.

Остаётся создать третий слой белого цвета, который будет создавать полоски на спасательном круге. Пока что сделаем его со сплошной обводкой.

## Спасательный круг, часть 4 [18/20]

Теперь все три «слоя» нашего рисунка готовы, и дело остаётся за малым.

Разрядим белую обводку последнего круга с помощью свойства stroke-dasharray, которое мы разбирали в [одном из прошлых заданий](https://htmlacademy.ru/courses/187/run/12).

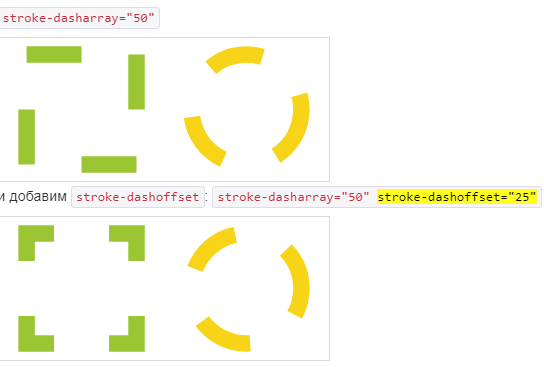
## Сдвиг обводки [19/20]

Пунктирной обводке с помощью свойства stroke-dashoffset можно задать сдвиг. Возьмём такой пример:

stroke-dasharray="50"

и добавим stroke-dashoffset: stroke-dasharray="50" stroke-dashoffset="25"

Пунктиры обводки сдвинулись на заданный отрезок по направлению против часовой стрелки. Значение stroke-dashoffset может быть отрицательным, и тогда обводка будет смещаться по часовой стрелке.



# Знакомство с HTML5

## Хедер и футер. Теги header и footer

В этом вводном курсе мы познакомимся с новыми возможностями HTML5. Основной упор сделаем на новые теги, но коснёмся и других нововведений.

Мы будем пошагово собирать блог известного вам веб-разработчика, используя появившиеся в HTML5 теги и применяя другие новшества. Но сначала несколько важных фактов об HTML5:

* HTML5 обратно совместим с HTML4.
* HTML5 не цельная спецификация, а набор модулей разной степени готовности.
* Многие модули HTML5 имеют хорошую поддержку во всех текущих браузерах.
* HTML5 с нами надолго.

Это означает, что HTML5 можно использовать уже сейчас. Для этого достаточно задать такой тип документа:

<!DOCTYPE html>

Что вы уже давным-давно делаете в наших курсах.

А теперь пара новых тегов:

* <header> — хедер сайта или раздела;
* <footer> — футер сайта или раздела.

Да-да, это те самые хедер и футер, которые обычно верстались дивами с классами header или footer. Они были настолько распространены, что для них было решено создать собственные теги.

## Основное содержание. Тег main

Обратите внимание, что мы задали классы для хедера и футера. Сделано это потому, что этих элементов на странице может быть несколько.

Хедер — это не только привычная шапка сайта с логотипом и меню, он может использоваться и как «шапка» какой-нибудь статьи или раздела сайта. Конечно, в случае со статьёй хедер называют не «шапкой», а вводной частью, в которой могут содержаться заголовки, оглавление и так далее.

С футером ситуация аналогичная. В привычном нам понимании это подвал сайта, с копирайтами, контактной информацией и так далее. Но футер может использоваться и в других разделах сайта. Например, в статье в футере можно разместить дополнительную информацию: данные об авторе, дополнительные ссылки и так далее.

А раз теги неуникальные, то и стилизовать их лучше с помощью классов, как мы и сделали.

Если вы не хотите использовать классы для шапки и подвала сайта, то можете использовать селекторы body > header и body > footer. Эти селекторы не повлияют на хедеры и футеры, вложенные более глубоко.

Другое дело новый тег <main>, который обозначает основное содержание сайта и по спецификации может использоваться на странице только один раз. Для его стилизации никакие классы не нужны.

Кстати, загляните в CSS и посмотрите, как сделаны фоны для основных блоков. В хедере и футере мы использовали линейные градиенты, которые детально разберём в более позднем курсе. А фоновую картинку для основного содержания мы задали с помощью так называемого data:URI, закодировав изображение прямо в CSS-коде. Это одна из продвинутых техник оптимизации вёрстки.

## Разделы страницы. Теги article и section

С шапками, подвалами и основным содержанием разобрались. Теперь перейдём к остальным разделам сайта.

Раньше почти все разделы верстались на дивах. Но в HTML5 добавили сразу два новых тега для разметки разделов:

* <section> — смысловой или логический раздел документа;
* <article> — самостоятельный и независимый раздел документа.

Чтобы не было путаницы, разберём где и когда использовать разные контейнеры:

* <div> — контейнер общего назначения, не обязательно смысловой. Дивы используются для разметки мелких блоков, создания сетки и декоративных эффектов.
* <section> — более крупный логический контейнер, объединяющий содержание по смыслу. Например, блок «О компании», список товаров, раздел личной информации в профиле и так далее.
* <article> — самостоятельный, цельный и независимый раздел документа. Этот раздел можно в неизменном виде использовать в различных местах, в том числе и на других сайтах. Примеры: статья, пост в блоге, сообщение на форуме и так далее.

В этом задании мы сверстаем блок «Обо мне» как независимый раздел, который можно будет использовать в разных местах сайта. А список постов — это логический раздел, группирующий последние посты, поэтому для него используем <section>.

## Изображения в формате SVG

Мы разметили базовые разделы нашей страницы. Теперь поработаем над каждым из них и начнём с шапки сайта.

Давайте добавим нашему сайту логотип. Для этого добавим в хедер <div>, а внутри него разместим привычное изображение. Используем картинку в векторном формате SVG, который уже достаточно хорошо поддерживается браузерами. Векторные изображения можно уменьшать и увеличивать без потери качества, в этом вы убедитесь в задании.

Кстати, обратите внимание на полупрозрачный фон в блоке «Обо мне». Он сделан с помощью нового формата записи цветов в CSS — rgba.

## Навигация. Тег nav

Теперь давайте добавим в шапку главное меню.

В HTML5 добавили специальный элемент для навигационных блоков — тег <nav>. Так что теперь меню и блоки ссылок лучше оборачивать не в дивы, а в навы.

В остальном же для меню ничего не изменилось. Это был список ссылок внутри дива, а стал список ссылок внутри нава.

## Завершаем футер

Теперь перейдём к доработке подвала сайта. Там нет ничего сложного. Просто нужно добавить пару смысловых разделов.

Обратите внимание на то, как повернётся изображение автора блога. Это сделано с помощью CSS-трансформаций, которые мы детально разберём в одном из последующих курсов.

## Использование нестандартных шрифтов

Перед тем как продолжить работу над сайтом, давайте разберёмся со шрифтами.

В HTML5 появилась возможность подключать и использовать на странице любые нестандартные шрифты. Веб-шрифты поддерживаются большинством современных браузеров.

Простейший способ использовать такой шрифт — найти его в специальном сервисе по названию, получить там код подключения шрифта, вставить этот код в свою вёрстку и использовать шрифт, как обычно, с помощью свойства font-family.

Подобных сервисов с бесплатными шрифтами достаточно много. Один из самых известных — это [Google Fonts](https://www.google.com/fonts). А вот похожий сервис: [fontstorage](https://fontstorage.com/ru/).

При использовании веб-шрифтов не забывайте указывать так называемые «фоллбэчные» шрифты — стандартные шрифты, которые будут отображаться, если веб-шрифт либо недоступен, либо не поддерживается старым браузером пользователя. Для этого нужно всего лишь перечислить их через запятую после нестандартного шрифта:

font-family: "PT Sans", "Arial", sans-serif;

Потренируемся подключать и использовать веб-шрифты.

## Подробнее о шрифтах. Правило @font-face

Технически подключение веб-шрифтов производится с помощью CSS-правила @font-face. Читается как «эт-правило font-face». Вот пример:

@font-face {

font-family: "Roboto";

src:

local("Roboto Regular"),

url("roboto.woff") format("woff");

}

В этом правиле вы указываете название шрифта, которое будете использовать в font-family и источники, из которых браузер сможет загрузить шрифт. Обычно сначала указывают название шрифта в системе, чтобы браузер попытался найти его локально, а затем указывают адрес файла шрифта в интернете.

Более подробно о технических тонкостях подключения шрифтов можно прочитать в серии статей на webfont.ru: [первая часть](http://webfont.ru/blog/about-font-face-part-one/) и [вторая часть](http://webfont.ru/blog/about-font-face-part-two/).

Получается, что можно хранить шрифты и подключать их со своего сервера. Это особенно полезно, когда шрифт очень редкий и его нет ни в одном из шрифтовых сервисов. В этом случае поступают так:

1. Берут файл шрифта (например, .ttf) и конвертируют в веб-формат в сервисе наподобие [Font Squirrel](http://www.fontsquirrel.com/tools/webfont-generator) (для кириллических шрифтов надо указать дополнительные параметры конвертации).
2. Затем сконвертированные файлы шрифта размещают у себя на сервере.
3. И подключают шрифт с помощью @font-face.

Сейчас мы подключим шрифт, который расположен на сервере htmlacademy. Мы уже сконвертировали шрифт в веб-формат.

## Дополнительное содержание. Тег aside

Теперь продолжим работу над страницей блога. Детальнее разметим раздел «Обо мне».

Этот раздел состоит из двух частей. В первой части будет содержаться основная информация, а во второй вспомогательная. Поэтому основную часть мы разметим с помощью уже знакомого <section>. А вторую часть разметим с помощью нового тега — <aside>.

<aside> — это дополнительное содержание, не связанное напрямую с основным. Ещё такие блоки часто называют «сайдбарами» или боковыми панелями.

И снова мы видим, как для общепринятого и часто используемого блока в HTML5 создали собственный элемент, немного расширив его значение и область применения.

## Ещё раз про article. Анонс поста

Теперь добавим пост в блок с последними постами.

Так как пост — это цельный, законченный и самостоятельный фрагмент информации, то для его разметки идеально подойдёт тег <article>.

Так как <article> должен быть независимым и легко встраиваемым куда угодно, то при разметке удобно считать его отдельной и самостоятельной страницей сайта.

Это означает, что у нашего поста будет своя собственная структура (шапка, подвал, разделы) и иерархия заголовков, которая будет начинаться с заголовка первого уровня.

Собственная иерархия заголовков у <article> — это очень удобная вещь, так как нам не нужно больше задумываться об окружении поста и подстраивать уровень заголовков в посте под это окружение, как часто приходится делать сейчас.

## Внутренняя страница: структура поста

В этом задании мы начинаем работу над внутренними страницами блога, то есть над самими постами.

Первым делом нам нужно разметить базовую структуру поста. Она будет похожа на ту, которую мы делали в 10 задании. Но будут и небольшие различия, ведь на главной странице был анонс поста, а здесь пост целиком.

Как вы уже догадались, в качестве контейнера поста мы используем <article>. У каждого поста будет шапка с заголовком и датой публикации, а также раздел введения и раздел с основным содержанием. Получается, внутри поста будут один <header> и два <section>.

Давайте договоримся для краткости называть <article> словом «статья», а <section>словом «раздел»

## Даты для людей и машин. Тег time

В HTML5 добавили специальный тег для обозначения даты и времени — <time>. У этого тега есть атрибут datetime — для указания даты в машиночитаемом формате ISO 8601.

С помощью <time> можно описывать даты одновременно и для человека, и для машины, например:

<time datetime="2014-04-20">Вчера</time> мы готовили курс к публикации.

Браузер отображает только содержимое тега, а содержимое datetime не отображается. Человек увидит только слово «вчера», а машина прочитает атрибут и получит дату в нужном ей формате. И все довольны.

Кстати, в этом задании вы не только добавите дату, но и наполните пост настоящим содержанием.

Раньше в спецификации указывалось, что у тега <time> может быть ещё один атрибут — pubdate. Сейчас этот атрибут из спецификации убран, и использовать его больше не нужно.

## Картинки с подписями. Теги figure и figcaption

Теперь, когда шаблон поста готов, можно добавить в блог ещё несколько коротких записей. А начнём с фотопоста.

Один из новых тегов — это <figure>.

Вспомните в книжках или учебниках изображения-выноски с подписями. Вот это и есть аналог <figure>. Цельный и независимый блок содержания. Внутри этого тега размещают демонстрационный материал: изображения, схемы, куски кода и так далее.

Обычно каждый такой материал сопровождает разъясняющий комментарий или «легенда». Для обозначения этого комментария и предназначен ещё один новый тег — <figcaption>, который размещается первым или последним элементом внутри <figure>. Пример:

<figure>

схема,

график,

диаграмма

и так далее

<figcaption>Легенда</figcaption>

</figure>

Мы используем этот тег для более прозаичных целей.

## Видео. Тег video

В HTML с самого начала было очень просто вставлять изображения, но очень сложно видео и аудио. В HTML5 эту проблему попытались решить, добавив специальные теги.

Для вставки видео предназначен тег <video>. Его основные атрибуты:

|  |  |
| --- | --- |
| width иheight | задают ширину и высоту видео |
| controls | пустой атрибут, при наличии которого отображается панель управления видео |
| preload | задаёт режим предзагрузки видео, имеет 3 возможных значения:   * none — не загружать ничего; * metadata — загрузить служебную мета-информацию (длительность, первый кадр и так далее); * auto — можно загрузить всё видео.   значение по умолчанию зависит от браузера |
| src | задаёт адрес видеофайла |
| autoplay | пустой атрибут, при наличии которого воспроизведение видео начинается автоматически |
| poster | задаёт адрес картинки-обложки, которая отображается, когда видео ещё не загрузилось или не воспроизводится |

## Форматы и источники видео

У тега <video> есть атрибут src, в котором можно указать адрес видеофайла, но мы почему-то его не использовали.

Всё дело в том, что в текущий момент существует несколько форматов видео, каждый из которых хорошо поддерживается лишь некоторыми браузерами. Вот три самых распространённых формата и их поддержка:

* [MPEG-4/H.264](http://caniuse.com/mpeg4)
* [OGG/Theora](http://caniuse.com/ogv)
* [WebM](http://caniuse.com/webm)

Поэтому мы должны в видео указывать адреса файлов во всех этих форматах (и конвертировать исходное видео в эти форматы, конечно). Делается это с помощью тегов <source>:

<video controls>

<source src="video.mp4" type="video/mp4">

<source src="video.ogv" type="video/ogg">

<source src="video.webm" type="video/webm">

</video>

В атрибуте src указывается адрес видеофайла, а в атрибуте type его тип (также там могут указываться и кодеки). Браузер из списка видеофайлов выбирает первый, который может проиграть и загружает его.

Атрибут type не является обязательным, так как браузер умеет сам определять тип и кодеки, но указывая тип явно, мы помогаем ему не ошибиться.

## Аудио. Тег audio

Работа с аудио в HTML5 очень похожа на работу с видео, только у тега для аудио чуть меньше параметров.

Для вставки звука предназначен тег <audio>. Его основные атрибуты:

|  |  |
| --- | --- |
| controls | пустой атрибут, при наличии которого отображается панель управления проигрывателем |
| preload | задаёт режим предзагрузки аудио, имеет 3 возможных значения:   * none — не загружать ничего; * metadata — загрузить служебную мета-информацию; * auto — можно загрузить весь файл.   значение по умолчанию зависит от браузера |
| src | задаёт адрес аудиофайла |
| autoplay | пустой атрибут, при наличии которого воспроизведение звука начинается автоматически |

## Форматы и источники звука

У звуковых файлов с поддержкой форматов дела обстоят лучше, чем у видео.

Для охвата большинства современных браузеров, достаточно использовать всего два формата:

* MP3
* OGG

И снова мы не можем указать только один файл в атрибуте src у тега <audio>. Мы должны так же, как и в случае с видео, перечислить адреса звуковых файлов в разных форматах с помощью тегов <source>:

<audio controls>

<source src="sound.mp3" type="audio/mpeg">

<source src="sound.oga" type="audio/ogg">

</audio>

Теги <source> в аудио работают так же, как и в видео.

# Формы и HTML5

## Простая кнопка

Помимо кнопок отправки формы или сброса введенных значений, существуют и «просто кнопки». При нажатии на такую кнопку никаких действий не происходит, а все необходимые действия обычно задаются при помощи JavaScript.

Пример записи:

<input type="button" value="Кнопка">

## Сброс введенных значений

Сначала разберёмся с возможностями кнопок, не рассмотренными в базовом курсе про формы.

Экспериментировать будем над формой логина в котопрофайл.

В HTML-формах есть специальная кнопка, которая сбрасывает введённые значения и возвращает изначально установленные. Это поле ввода с типом reset.

Пример использования:

<input type="reset" value="Сбросить">

Обратите внимание, что кнопка не обнуляет значения, а возвращает те, которые были установлены в полях формы по умолчанию.

## Кнопка-изображение

В качестве кнопки отправки формы можно использовать изображение.

Для этого у тега input нужно указать тип image.

Аналогично обычным изображениям на сайте у кнопки-изображения есть еще два атрибута:

|  |  |
| --- | --- |
| src | адрес изображения |
| alt | альтернативный текст, отображаемый в том случае, если изображение не загружено |

Кнопка-изображение работает аналогично кнопке submit, но на сервер дополнительно передаются координаты точки, по которой был произведен щелчок.

## Альтернативный способ задания кнопок

Помимо тега <input> для добавления кнопок можно использовать тег <button>. Он расширяет возможности создания кнопок.

Внутри тега <button> можно размещать любые HTML-элементы, в том числе изображения. Например:

<button>Календарь <img src="calend.png" alt=""></button>

В данном случае кнопка будет выглядеть примерно вот так: Кнопка

Если в атрибуте type тега <button> указать значение submit или reset, то кнопка будет отправлять данные на сервер или сбрасывать введенные значения.

По умолчанию значение атрибута type — submit.

То есть внутри формы кнопка <button> по нажатию отправит форму на сервер.

Значениями атрибута type также могут быть button и reset.

Кнопка button со значением type="reset" аналогично input type="reset" сбрасывает значения полей формы к изначальным.

А вот значение type="button" избавит кнопку от всей изначальной фунциональности. То есть кнопка просто будет выглядеть как кнопка, но ничего по умолчанию не делать. Это удобно, если вы хотите сами добавить кнопке дополнительное действие с помощью JavaScript.

## Обязательные поля

Ура, товарищи! Мы успешно разобрались с возможностями кнопок в формах и залогинились в профайл Кексика.

Теперь нам предстоит изучить новые возможности форм, большая часть которых была добавлена в HTML5. А в процессе мы будем помогать Кексику строить формы, используя полученные знания.

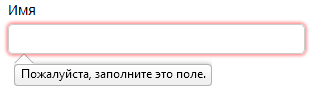
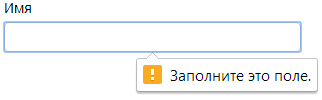
Итак, начнём...

Кексик, как и любой уважающий себя кот, любит рыбу. Но при этом каждый раз ходить в рыбную лавку за лососем ему лень. Поэтому он, как уважающий себя инженер, решил облегчить свою жизнь автоматизацией процесса: он собрался сделать форму доставки любимых лакомств на дом, чтобы ей мог пользоваться он сам и его друзья-котаны. Кексик начал её верстать, и ему надо помочь в этом нелёгком деле.

Сначала давайте разберёмся с обязательными полями. Чтобы указать, что поле обязательно для заполнения, нужно добавить ему пустой атрибут required:

<input type="text" required>

При попытке отправить форму с незаполнеными обязательными полями браузер выведет всплывающее предупреждение, которое в Chrome и Firefox выглядит вот так:



Эта проверка работает на клиентской части и упрощает валидацию форм.  
Но всегда нужно проверять отправленные данные и на стороне сервера.

## Поле выбора даты

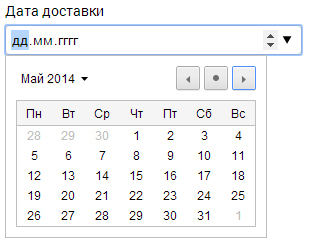
Поля для задания даты и времени уже поддерживаются в [этих браузерах](http://caniuse.com/#feat=input-datetime).

В форму заказа важно включить поле даты доставки. А для выбора даты из календаря существует новый тип поля ввода — date. При клике на данное поле в форме всплывает календарик.

Пример записи:

<input type="date">

В Chrome это выглядит так:



Если браузер не поддерживает поле для ввода даты, то вместо него отображается обычное текстовое поле.

## Поле выбора времени

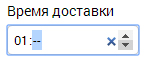
Поля для задания даты и времени уже поддерживаются в [этих браузерах](http://caniuse.com/#feat=input-datetime).

В предыдущем задании мы рассмотрели использование тега <input> с типом date. Но иногда нужно указывать дату в других форматах, и для этого существуют дополнительные «временные» типы полей, например, time для выбора времени.

Используем поле выбора времени в нашей форме:

<input type="time">

В Chrome это выглядит так:



Если браузер не поддерживает поле для ввода времени, то вместо него отображается обычное текстовое поле.

## Список возможных значений

Списки возможных значений уже поддерживаются в [этих браузерах](http://caniuse.com/#feat=datalist).

Для текстовых полей можно заранее определить список возможных значений, которые отображаются, когда вы начинаете вводить текст в поле. Для этого существует специальный тег <datalist>.

Пример использования:

<input type="text" list="browsers" name="browser">

<datalist id="browsers">

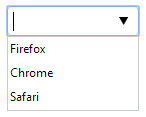
<option value="Firefox">

<option value="Chrome">

<option value="Safari">

</datalist>

В Chrome это выглядит так:



Связывание текстового поля и списка осуществляется при помощи атрибута list у тега input — значение list должно быть таким же, как значение атрибута id у списка.

Если тег input имеет специфический тип, например email или другие, которые будут рассмотрены далее в курсе, то в списке отображаются только корректные для данного типа значения.

## Поле ввода числового значения

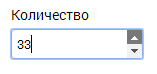
Поля для ввода числовых значений уже поддерживаются в [этих браузерах](http://caniuse.com/#input-number).

Для ввода числовых значений существует специальный тип поля ввода number. Рядом с полем браузер автоматически подставляет две стрелочки для увеличения и уменьшения числового значения.

Пример записи:

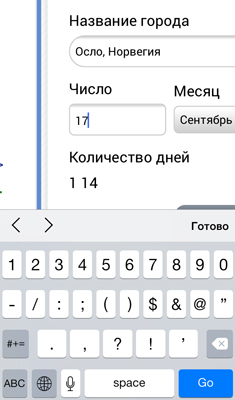
<input type="number">

В Chrome это выглядит так:



При помощи вспомогательных атрибутов min и max можно установить верхнюю и нижнюю границу допустимых значений. А атрибут step устанавливает величину шага изменения значения.

Также стоит отметить, что поле ввода числа, как и некоторые другие поля, которые будут рассмотрены дальше, по-особому ведёт себя в мобильных браузерах: например, при фокусе на такое поле появляется клавиатура, позволяющая вводить соответствующие символы.



## Поле поиска

Отлично! Мы справились с формой заказа и можем смело двигать дальше.

Кексик давно намеревался побывать в Норвегии и отведать хвалёной местной рыбки. Чтобы упростить себе жизнь и не ходить в турфирму, он решил создать страницу с формой бронирования отеля.

И тут без вашей помощи никак не обойтись.

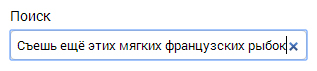
Первым делом в форму нужно поместить поле поиска, чтобы можно было напечатать название города и потом выбирать подходящие отели.

Пример записи:

<input type="search">

Это поле почти не отличается от обычного текстового поля. В некоторых браузерах внутри него появляется крестик для сброса введённого значения.

В Chrome это выглядит так:



## Автофокус

Автофокус уже поддерживается в [этих браузерах](http://caniuse.com/#feat=autofocus).

При загрузке страницы можно сообщить браузеру в какое поле установить курсор по умолчанию. Для этого используется пустой атрибут autofocus.

Пример записи:

<input type="text" autofocus>

Автофокус улучшает процесс работы с формами, ведь пользователь избавляется от лишних щелчков мышки, там где они не нужны, а может сразу начинать вводить текст в поле.

Обратите внимание, что такой атрибут должен быть только один на странице.

## Другие поля для ввода дат

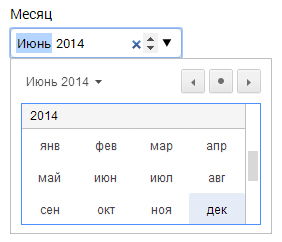
Поля для задания даты и времени уже поддерживаются в [этих браузерах](http://caniuse.com/#feat=input-datetime).

В предыдущих заданиях мы уже использовали поля с типами date и time, но существуют и другие типы полей для ввода дат:

|  |  |
| --- | --- |
| datetime | выбор даты с указанием времени (c учетом временной зоны) |
| datetime-local | выбор даты с указанием времени (без учета временной зоны) |
| week | выбор порядкового номера недели в году и года |
| month | выбор месяца и года |

Используем поле для выбора месяца в нашей форме. А для выбора числа используем уже знакомый тип number.

Вот так выбор месяца выглядит в Chrome:



Если браузер не поддерживает поле для ввода месяца, то вместо него отображается обычное текстовое поле.

## Выбор из диапазона

Поле для выбора из диапазона значений уже поддерживаются в [этих браузерах](http://caniuse.com/#feat=input-range)

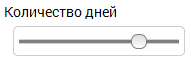
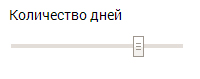
В нашей форме обязательно должна быть возможность выбрать количество дней, на которое будет осуществляться бронирование. Для этой цели подходит тип поля range.

Такое поле выглядит как шкала с ползунком и позволяет выбрать число из некоторого интервала значений.

Пример записи:

<input type="range" min="1" max="10">

В Chrome и Firefox это поле выглядит так:



Атрибуты min и max устанавливают нижнюю и верхнюю границу допустимых значений. А атрибут step устанавливает величину шага изменения значения.

## Область для вывода результата

Область для вывода результата уже поддерживаются в [этих браузерах](http://html5test.com/compare/feature/form.output.element/form-output-element.html)

Чтобы видеть количество дней в брони, используем новый HTML5-тег <output>.

Тег <output> представляет собой область, куда выводятся какие-либо результаты вычислений, обычно полученные при помощи JavaScript.

Пример записи:

<output name="sum">[значение по умолчанию]</output>

Значение по умолчанию при этом можно не задавать, тогда область вывода будет пустой.

В этом задании мы добавим в форму область вывода, в которой будет отображаться количество дней, выбранное с помощью поля-ползунка.

## группировка полей формы

Когда формы становятся очень большими, возникает потребность зрительно отделить одни поля от других. Для группировки полей используется тег fieldset.

Пример:

<fieldset>

<input type="text">

<input type="text">

<input type="text">

</fieldset>

<fieldset>

<textarea></textarea>

</fieldset>

По умолчанию браузеры отображают результат в виде рамки вокруг этой группы полей, но при помощи CSS можно изменить его внешний вид.

Также для каждой группы можно добавить её заголовок. Для этого внутрь тега fieldsetнадо поместить тег legend:

<fieldset>

<legend>Заголовок группы</legend>

<input type="text">

</fieldset>

## Паттерны значений полей

Паттерны значений полей уже поддерживаются в [этих браузерах](http://caniuse.com/#feat=input-pattern).

Итак, у нас готова форма бронирования отеля, но чтобы отправиться в Норвегию, Кексику нужно получить визу.

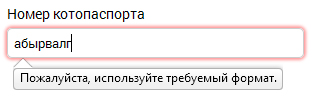
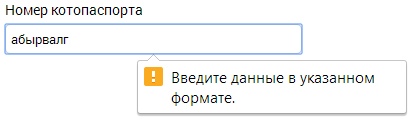
Естественно, ему лень идти в консульство и лично предъявлять там усы, лапы и хвост, поэтому он решает создать ещё одну форму, которая сможет отправить все необходимые данные по получению визы прямиком в лапы норвежского посла.

Ни для кого не секрет, что основным кошачьим документом является котопаспорт, в котором зафиксированы окрас кота, его личные данные и гастрономические предпочтения.

Поэтому первым делом добавим в форму поле для номера котопаспорта. Это обычное текстовое поле, но с ограничениями на формат номера, чтобы не злить лишний раз норвежского посла ошибками и опечатками.

Чтобы добавить автоматическую проверку формата номера в поле, используем атрибут pattern, в котором с помощью регулярного выражения опишем требуемый формат.

В Chrome и Firefox подобное поле с неправильно введёнными данными выглядит так:



Подробнее о регулярных выражениях можно почитать, например, [на Википедии](http://ru.wikipedia.org/wiki/%D0%A0%D0%B5%D0%B3%D1%83%D0%BB%D1%8F%D1%80%D0%BD%D1%8B%D0%B5_%D0%B2%D1%8B%D1%80%D0%B0%D0%B6%D0%B5%D0%BD%D0%B8%D1%8F).

Добавьте этому полю атрибут pattern со значением  
[0-9]{3}-[0-9]{5}

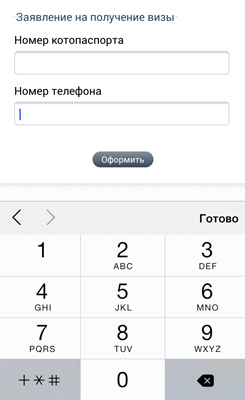
## Поле ввода телефона

Для получения визы нужно сообщить и номер телефона Кексика, поэтому добавим в форму ещё одно поле с типом tel.

Новый тип поля tel появился в HTML5 и отвечает за ввод телефонных номеров.

Также воспользуемся атрибутом pattern, чтобы исключить ошибки при заполнении формы.

В мобильных браузерах при фокусе на такое поле появляется клавиатура, позволяющая вводить только цифры и символы телефонных номеров.



## Подсказка при заполнении полей

У полей, в которые вводятся текстовые значения (textarea, разные типы input и так далее) есть возможность вывести подсказку.

Для этого используется специальный атрибут placeholder:

<input type="text" placeholder="Текст подсказки">

Текст подсказки выводится внутри текстового поля, а при вводе значения — автоматически убирается.

Воспользуемся этим атрибутом, чтобы подсказывать в каком именно формате следует вводить номера котопаспорта и телефона.

## Поля ввода адресов сайтов и email

Вот поддержка браузерами [полей для ввода email](http://html5test.com/compare/feature/form.email.element/form-email-validation.html) и [полей для ввода адресов сайтов](http://html5test.com/compare/feature/form.url.element/form-url-validation.html)

В HTML5 добавлены два типа полей email и url, предназначенные для ввода электронной почты и адреса сайта. Особенностью этих полей является то, что они автоматически проверяют формат введённых данных.

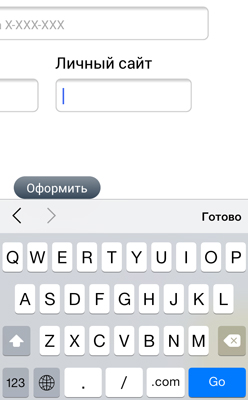
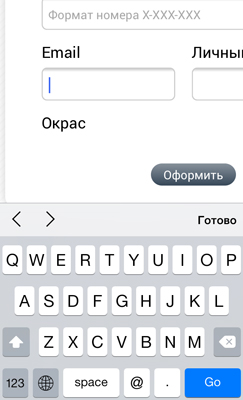
Пример записи:

<input type="email">

<input type="url">

Внешне эти поля не отличаются от обычных текстовых полей, но обладают важной особенностью, которая очень полезна на мобильных устройствах.

Когда вы начинаете заполнять такое поле на мобильнике, там автоматически переключается раскладка клавиатуры. Например, для email отобразятся латинские символы, цифры, знак @ и некоторые другие. Посмотрите на скриншоты:



## Поле выбора цвета

Поля для выбора цвета уже поддерживаются в [этих браузерах](http://caniuse.com/#feat=input-color)

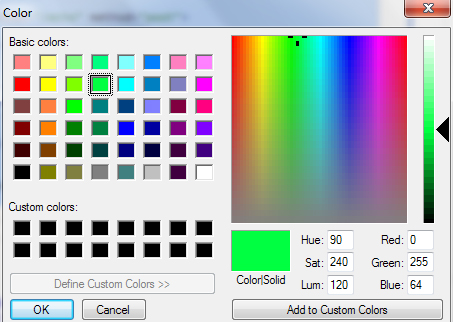
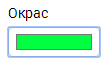
Нельзя упускать из виду важный параметр кота — окрас. Его также нужно ввести в нашей форме.

В HTML5 добавили новый тип color, предназначенный для полей выбора цвета. При клике на такое поле появляется окно с возможностью выбрать цвет из палитры.

Пример записи:

<input type="color">

В Chrome это выглядит так:



Если браузер не поддерживает поле для выбора цвета, то вместо него отображается обычное текстовое поле.

## Группировка элементов списка

И последнее по очерёдности, но очень важное, поле в нашей форме — поле выбора гастрономических предпочтений Кексика.

Так как список выбора может быть довольно большой, в теге select используем возможность объединять option в группы. Обычно это используется для большей наглядности и удобства поиска нужного варианта.

Для формирования группы используется тег optgroup. Атрибут label этого тега определяет заголовок группы.

Пример использования:

<select name="variants">

<optgroup label="Группа вариантов 1">

<option value="1">Вариант 1</option>

<option value="2">Вариант 2</option>

<option value="3">Вариант 3</option>

</optgroup>

<optgroup label="Группа вариантов 2">

<option value="4">Вариант 4</option>

<option value="5">Вариант 5</option>

<option value="6">Вариант 6</option>

</optgroup>

</select>

Вложенность групп не ограничена, внутрь каждой группы можно вложить другие группы.

Аналогично можно группировать элементы и в списках со множественным выбором.

## Запрет редактирования полей

Иногда возникают ситуации, когда какие-то поля требуется сделать недоступными для редактирования.

Есть два способа: использование атрибута readonly и использование атрибута disabled

Пример записи:

<input type="text" readonly>

<input type="text" disabled>

В чем же отличие между ними?

Атрибут readonly не дает пользователю изменять поле (вводить новый текст, модифицировать существующий). Введенное значение можно выделить и скопировать. Данные из этого поля отправляются на сервер.

Атрибут disabled не дает пользователю изменять поле (вводить новый текст, модифицировать существующий). Нельзя поставить фокус в это поле, введенное значение нельзя выделять и копировать. Данные из этого поля НЕ отправляются на сервер.

## Управление автозаполнением полей

Браузер может запоминать значения, вводимые в текстовые поля. При вводе первых букв текста выводится список сохранённых ранее значений, из которого можно выбрать подходящее. Параметрами автозаполнения можно управлять используя атрибут autocomplete.

Он может принимать два значения on и off. Первое включает автозаполнение, второе — отключает. Отключение автозаполнения обычно используется из соображений безопасности, например, чтобы не сохранялись пароли, номера банковских карт и так далее.

Пример использования:

<input type="text" autocomplete="off">

Значение по умолчанию зависит от настроек браузера.

## Переключение между полями

При нажатии клавиши Tab браузер передает управление (фокус) от одного элемента к другому в том порядке, в котором они были объявлены на странице. Этим порядком можно управлять при помощи атрибута tabindex.

Пример записи:

<input type="text" tabindex="3">

В качестве значения может использоваться любое целое положительное число. Значения выстраиваются последовательно и переход между элементами происходит от меньшего значения к большему.

Если представлено отрицательное значение — элемент может быть выделен, однако не учавствует в последовательной навигации

Если представлен 0 — элемент может быть выделен и достигнут с помощью последовательной навигации, однако порядок навигации определён платформой

Из соображений доступности не рекомендуется менять порядок навигации по полям ввода по умолчанию.

## localStorage

Кексик наконец-то попал в Норвегию и во время путешествия начал вести путевые заметки. Даже создал для этого простую форму. Но из-за плохого интернета форму часто не удавалось отправить и введённая заметка терялась.

Как справиться с этой довольно часто встречающейся проблемой, когда введённые в форму данные теряются?

Для этого можно использовать одну из новинок HTML5 — localStorage. Эта технология относится к JavaScript, а не к HTML или CSS, но промолчать о ней мы не можем.

Суть localStorage или «локального хранилища» заключается в том, что в него можно записывать данные, которые будут сохраняться в браузере. Эти данные не исчезнут даже если вы закроете браузер и откроете его снова или уйдете со страницы и потом вернётесь на неё.

Получается, что можно сохранять данные из формы в хранилище при работе с формой, а при загрузке страницы проверять хранилище на наличие данных, и если они есть, то подставлять их в форму. Таким образом можно предотвратить потерю данных при работе с формами.

Давайте проверим работу localStorage. Это просто. Мы уже всё подготовили, вам осталось только раскомментировать один тег.

## Проверяем работу localStorage

На прошлом шаге вы раскомментировали скрипт, который сохранил данные из формы в локальное хранилище вашего браузера.

На этом шаге мы проверим, что данные действительно сохранились.

Для этого нужно раскомментировать тот же самый скрипт, что и в предыдущем шаге. И если у вас достаточно современный браузер, который поддерживает localStorage, то вы увидите в форме текущего задания ваш текст.

Заметьте, что в исходном коде задания тег <textarea> пуст, поэтому данные в форму могут попасть только из хранилища. Можете даже провести эксперимент: закрыть браузер, запустить его, открыть это задание, раскомментировать скрипт и убедиться, что сохранённый текст появится вновь.

Подробнее о localStorage и других возможностях HTML5, которые позволяют улучшить опыт взаимодействие с формами, вы можете прочитать в [одноимённой статье](http://simonenko.su/38146501854/improving-ux-for-web-form) в блоге Алексея Симоненко.

# Селекторы, часть 2

## Объединение селекторов

Иногда может потребоваться выбрать элементы, которые одновременно удовлетворяют сразу нескольким условиям.

В CSS есть запись, которая фактически выполняет операцию логического умножения, «И». Селекторы, применяемые к одному элементу, в этом случае пишутся без пробелов:

.class1.class2 { }

Стили будут применяться ко всем элементам, которые одновременно имеют класс class1и class2. Ведь это же не новость для вас, что HTML-элементы могут одновременно иметь несколько классов. Например:

<div class="class1 class2">Блок с двумя классами</div>

По такому же принципу можно объединять любое количество абсолютно разных селекторов. И чем больше селекторов вы объединяете, тем больше условий должно совпасть для применения стилей.

В программировании похожую операцию часто обозначают как &&. Например:

if (firstSelector && secondSelector) { ... }

## Псевдокласс :not

Псевдокласс :not(селектор) является отрицающим селектором. С его помощью можно выбрать элементы, которые НЕ содержат указанный селектор:

li:not(:last-child) { }

Этот селектор выберет все теги <li>, НЕ являющиеся последними в их родителе.

Псевдокласс :not похож на оператор ! в программировании:

if (!selector) { ... }

В качестве селектора могут указываться псевдоклассы, теги, идентификаторы, классы и селекторы атрибутов. Нельзя использовать двойной псевдокласс :not, то есть конструкция :not(:not(...)) не сработает.

Также в комбинации с :not не применяются:

* объединение селекторов: например, li:not(.heart.jack) — некорректный селектор;
* псевдоэлементы: li:not(::after) — неправильная запись (подробнее о псевдоэлементах рассказано [далее в курсе](https://htmlacademy.ru/courses/57/run/14));
* селекторы-потомки, групповые селекторы или комбинации: например, нельзя писать li:not(a span) или li:not(a + span).

## Комбинируем :not

Отрицающий селектор :not, как и любые другие селекторы, можно комбинировать с другими. Например:

li:not(:first-child):not(:last-child) { }

Выберет все теги <li>, которые НЕ являются первыми и последними в их родителе.

Объединять можно неограниченное количество селекторов.

## Псевдокласс :nth-last-child

В первом курсе про селекторы мы уже рассматривали псевдокласс :nth-child, сейчас посмотрим на :nth-last-child.

Псевдокласс :nth-last-child используется для добавления стиля к элементам на основе нумерации в дереве элементов. В отличие от псевдокласса :nth-child, отсчет ведется не от первого элемента, а от последнего. Вот и все различия.

Подробнее об :nth-child и синтаксисе выражений рассказано в пошаговой демонстрации [Использование псевдокласса :nth-child](https://htmlacademy.ru/demos/2)

## Псевдокласс :first-of-type

Псевдокласс :first-of-typeочень похож на :first-child. Он выбирает первый дочерний элемент родителя, только с учетом типа элементов.

Например, в этом задании перед списками с картами есть блок с текстом. У списков и у блока с текстом общий родитель — body.

<body>

<div class="paper"> … </div>

<ul class="cards"> … </ul>

<ul class="cards"> … </ul>

</body>

Сравним две записи:

ul:first-child {

background-color: #ffffee;

}

ul:first-of-type {

background-color: #ffffee;

}

Верхний селектор выбирает первый дочерний элемент в родителе, причем этот элемент должен быть ul. В нашем случае не выберется ничего, потому что первым дочерним элементом body является div.

Нижний селектор выбирает первый ul среди всех дочерних ul в своем родителе. В нашем случае будет выбрана первая строка с картами.

## Псевдокласс :last-of-type

В предыдущем задании мы рассмотрели псевдокласс :first-of-type.

Псевдокласс :last-of-type работает аналогично, только выбирает последний дочерний элемент родителя с учетом типа.

Пример записи:

ul:last-of-type {

...

}

## Псевдокласс :nth-of-type

Псевдокласс :nth-of-type работает почти так же, как и :nth-child. Разница заключается в том, что он учитывает тип элемента.

Вы можете обратиться к разбору примера [в пятом задании](https://htmlacademy.ru/courses/57/run/5), чтобы разобраться в отличиях этих селекторов.

Еще пример. Если в текущем задании мы используем такие селекторы:

ul:nth-child(2) { }

ul:nth-of-type(2) { }

То верхний cелектор выберет второй по счёту дочерний элемент и этот элемент должен быть ul. В нашем случае выберется первая строка карт.

А нижний селектор выберет второй по счету ul среди дочерних ul. В нашем случае выберется вторая строка карт.

Информация о синтаксисе :nth-child описана в задании [Псевдокласс :nth-child](https://htmlacademy.ru/courses/42/run/10) курса «Селекторы, часть 1».  
Вот неплохая [дополнительная статья](https://css-tricks.com/the-difference-between-nth-child-and-nth-of-type/) о различиях :nth-child и :nth-of-typeи её [перевод на Хабре](http://habrahabr.ru/post/119139/).

## Псевдокласс :nth-last-of-type

Наверняка, вы и сами уже прекрасно догадались, как будет работать :nth-last-of-type.

Элементы выбираются по их расположению, отсчет ведется от конца, учитывается тип элемента.

Пример записи:

ul:nth-last-of-type(2) { }

Когда набор элементов не очень большой, все эти псевдоклассы :nth-child, :nth-of-type, :nth-last-child, :nth-last-of-type можно легко заменить один другим.

На более сложных структурах зачастую бывает удобнее использовать только определённые из них. Например, когда нужно выделять второй элемент с конца, но количество элементов в списке изменяется.

## Cелектор последующих элементов

Ранее мы уже рассматривали [соседние селекторы](https://htmlacademy.ru/courses/42/run/6), которые записываются как селектор1 + селектор2.

Есть похожий селектор селектор1 ~ селектор2. Стили применятся к элементу, подходящему под селектор2, только если перед ним расположен элемент, подходящий под селектор1.

Отличие от соседнего селектора состоит в том, что между элементами селектор1и селектор2 могут находиться другие элементы. Поэтому будем называть селектор1 ~ селектор2 селектором следующих элементов.

Сравним:

<ul class="cards">

<li class="king diamond">

<li class="queen heart">

<li class="jack spade">

<li class="ace heart">

<li class="king club">

</ul>

.queen.heart + li {

background-color: #ffff99;

}

.king.diamond ~ li {

background-color: #99ddff;

}

В первом случае выделится одна карта, расположенная сразу за червовой дамой, то есть пиковый валет.

Во втором случае выделятся все карты, которые стоят за бубновым королем, то есть червовая дама, пиковый валет, червовый туз и трефовый король.

## Псевдокласс :empty

Псевдокласс :empty, выбирает только те теги, у которых нет дочерних элементов (в том числе текстовых узлов). Учтите, что даже переход на новую строку считается текстовым узлом, помните об этом в процессе проектирования структуры страницы.

Пример:

ul:empty {

...

}

В данном случае выберутся все пустые элементы ul.

## Псевдокласс :only-child

Псевдокласс :only-child пригодится, когда нужно прописать индивидуальные стили для элемента, который является единственным дочерним элементом внутри родительского контейнера.

Пример:

li:only-child {

...

}

Сработает, когда этот <li> в списке единственный.

Cелектор :only-child эквивалентен селектору elem:last-child:first-child.

## Псевдокласс :only-of-type

Псевдокласс :only-of-type работает почти так же, как и :only-child. Отличие состоит в том, что он учитывает тип элемента.

Пример:

p:only-of-type {

...

}

В данном случае стили будут применены к элементу p, если это единственный p внутри своего родителя.

## Псевдоэлемент ::before

Псевдоэлемент before позволяет с помощью CSS добавить псевдотег внутрь другого элемента и оформить его. Cодержимое псевдотега задаётся с помощью свойства content.

Например, у нас есть такой HTML:

<div class="queen heart">

<em>Дама</em>

</div>

Добавим CSS-правило с необычным селектором с двойным двоеточием:

.heart::before { content: "Черви"; }

И HTML изменится вот так:

<div class="queen heart">

*<before>Черви</before>*

<em>Дама</em>

</div>

Но! Исходный HTML-код не изменится, тег <before> не попадёт в код страницы, а будет «виртуально» существовать где-то в браузере. Поэтому мы и используем приставку псевдо.

Ведёт себя псевдотег так же, как обычный <span> с текстом. Ему можно задавать дополнительные стили, например:

.heart::before {

content: "Черви";

color: red;

}

Чтобы псевдоэлемент появился, ему необходимо задать свойство content. Достаточно даже пустой строки в значении свойства — content: "";.

## Псевдоэлемент ::after

Псевдоэлемент after аналогичен before. Отличие заключается в том, что он добавляет псевдотег не в начало, а в конец элемента. Например:

.heart::after { content: "Черви"; }

Даст такой результат:

<div class="queen heart">

<em>Дама</em>

*<after>Черви</after>*

</div>

Псевдоэлементы before и after можно использовать одновременно. Это означает, что с помощью CSS вы можете добавить к любому элементу на странице два псевдоэлемента.

Обратите внимание, что псевдоэлементы пишутся с двойным двоеточием. Этим они отличаются от псевдоклассов, которые используют одинарное двоеточие.

В старом варианте стандарта псевдоэлементы можно было использовать с :. Поэтому запись с : понимают и очень старые браузеры. Но сейчас использовать одинарное двоеточие для псевдоэлементов считается дурным тоном.

## Позиционирование псевдоэлементов

Вы можете задавать псевдоэлементам любые CSS-свойства. Можно менять тип элемента, задавать ему позиционирование, отступы, фон и так далее.

Этими возможностями особенно часто пользуются при создании различных декоративных эффектов. Это очень удобно.

Во-первых, не нужно добавлять лишний тег под каждую очередную мелкую иконку.

Во-вторых, можно управлять этими иконками/декоративными элементами только с помощью CSS, что открывает огромный простор для создания интересных динамических эффектов.

## Фон для псевдоэлементов

Наша карта почти готова. Осталось сделать последний штрих — задать фоны самой карте и псевдоэлементам.

Кстати, мы перевернули нижнюю иконку с помощью CSS-свойства transform, которое будет рассмотрено в курсе про [двумерные трансформации](https://htmlacademy.ru/courses/71).

Также для управления размером фонового изображения мы использовали свойство background-size, которое рассмотрим в курсе про [рамки и фоны](https://htmlacademy.ru/courses/88).

## Псевдоэлементы ::first-line и ::first-letter

Отвлечемся от карт и обратимся к тексту.

Псевдоэлемент first-line задает стиль первой строки форматированного текста. Длина этой строки зависит от многих факторов, таких как используемый шрифт, размер окна браузера, ширина блока, языка и так далее. В правилах стиля допустимо использовать только свойства, относящиеся к шрифту, изменению цвета текста и фона.

Пример использования:

p::first-line { }

Аналогично псевдоэлемент first-letter определяет стиль первого символа в тексте элемента, к которому добавляется. К этому псевдоэлементу могут применяться только стилевые свойства, связанные со свойствами шрифта, полями, отступами, границами, цветом и фоном.

Пример использования:

p::first-letter { }

# Курс «[Селекторы, часть 3](https://htmlacademy.ru/courses/58)»

## Ищем в начале строки: [foo^="bar"] [1/18]

В [первом курсе про селекторы](https://htmlacademy.ru/courses/42/) мы разбирали селектор по атрибутам, когда запись input[type="text"] выберет все элементы input, у которых атрибут type равен text.

У этого механизма есть дополнительные возможности: можно выбирать элементы по вхождению подстроки в значение атрибута.

Запись вида [foo^="bar"] выберет все элементы, у которых значение атрибута foo начинается с подстроки bar.

Представьте, что у вас есть три класса для задания колонок разной ширины, например: column-1, column-2 и column-3.

У этих классов часть свойств повторяется, а разной является только ширина. Чтобы не дублировать CSS-код, вы можете вынести общие свойства колонок в правило с селектором [class^="column-"], а в остальных правилах задать только ширину:

[class^="column-"] {

/\* общие свойства: отступы, рамки, фон и т.д. \*/

}

.column-1 { width: 100px; }

.column-2 { width: 200px; }

.column-3 { width: 300px; }

То есть, первый селектор выберет все дивы с классами, начинающимися на column-:

<div class="column-1"></div>

<div class="column-2"></div>

<div class="column-3"></div>

Обратите внимание, что селектор чувствителен к регистру.

## Ищем в конце строки: [foo$="bar"] [2/18]

Селектор вида [foo$="bar"] выбирает все элементы, значение атрибута foo которых оканчивается строкой bar.

Представьте, что у вас на сайте есть раздел с файлами для скачивания в разных форматах и вам нужно для каждого типа файлов добавить свою иконку. Пример разметки:

<a href="batman.pdf">Скачать</a>

<a href="superman.doc">Скачать</a>

В этом случае вы можете назначать иконки в CSS по расширениям файлов:

a[href$=".pdf"] {

/\* иконка для PDF \*/

}

a[href$=".doc"] {

/\* иконка для DOC \*/

}

И снова, селектор чувствителен к регистру.

## Поиск подстроки везде: [foo\*="bar"] [3/18]

Следующий вариант записи [foo\*="bar"]

Будут выбраны все элементы, у которых значение атрибута foo содержит подстроку bar на любой позиции

Среди трёх элементов:

<p class="person-name"></p>

<div class="some-person-info"></div>

<span class="date-person"></span>

селектор [class\*="person"] выберет все.

Обратите внимание, что селектор чувствителен к регистру.

Курс «[Селекторы, часть 3](https://htmlacademy.ru/courses/58)»

## Поиск слов внутри строки: [foo~="bar"] [4/18]

Следующая запись: [foo~="bar"].

Такой селектор выберет все элементы, у которых значение атрибута foo содержит слово bar.

Входить должно именно слово, а не просто подстрока. То есть вхождение bar должно содержать с обеих сторон разделители: пробелы или начало/конец строки.

Курс «[Селекторы, часть 3](https://htmlacademy.ru/courses/58)»

## Поиск префиксов: [foo|="bar"] [5/18]

Селектор по атрибутам вида [foo|="bar"]

В данном случае будут выбраны все элементы, у которых значение атрибута foo содержит префикс bar, то есть либо полностью совпадает с bar, либо начинается со строки bar-(наличие знака переноса существенно). Другими словами, используя уже знакомые записи селекторов, этот можно заменить на два:  
1. [foo="bar"] — все элементы, у которых значение атрибута foo полностью совпадает со значением bar.  
2. [foo^="bar-"] — все элементы, у которых значение атрибута foo начинается со значения bar-.

## Поиски котов. Часть 1 [6/18]

Наверно все уже устали от этих форм, поэтому немного сменим тему и займемся поисками котов.

Естественно, используя уже изученные новые селекторы.

В этот раз история совсем простая: из продуктового магазина выкинули старые коробки, и все коты с округи сбежались и залезли в них. Будем искать, кто в какой!

Ищем котов по значению атрибута title у элемента .cat-in-box.

Чтобы открыть коробку, необходимо у элемента .cat-in-box взять псевдоэлемент before и установить ему свойство display: none;, а чтобы закрыть — display: block;.

Объединяйте селекторы, чтобы найти нужную коробку.

## Поиски котов. Часть 2 [7/18]

Поиски котов продолжаются!

Ищем котов по значению атрибута title у элемента .cat-in-box

Чтобы открыть коробку необходимо у элемента .cat-in-box взять псевдоэлемент before и установить ему свойство display: none, а чтобы закрыть - display: block

Объединяйте селекторы, чтобы найти нужную коробку

## Псевдоклассы :enabled и :disabled [9/18]

В предыдущих курсах мы уже научились работать с формами и разными полями форм. Теперь рассмотрим ряд дополнительных селекторов для работы с этими элементами.

Для обращения к элементам, которые являются доступными на сайте (не заблокированными), можно использовать псевдокласс :enabled. Заблокированными считаются элементы форм, у которых установлен атрибут disabled. Подробнее об этом атрибуте можно посмотреть в [этом курсе](https://htmlacademy.ru/courses/74)

Например:

input:enabled {

/\* какие-то стили \*/

}

И наоборот, если нужно обратиться только к заблокированным элементам, то для этого есть псевдокласс :disabled

## Псевдоклассы :read-only и :read-write [10/18]

Как мы уже рассматривали в предыдущих курсах, есть разные способы запретить редактирование пользователем полей. Одним из таковых является установка атрибута readonly. Значение в данном случае доступно для чтения и копирования, но недоступно для редактирования.

В зависимости от этого параметра существует два селектора, которые позволяют выбирать каждую группу полей

Селектор :read-write выберет все поля доступные для редактирования

Селектор :read-only выберет все поля доступные только для чтения

Пример записи:

input:read-only {}

В случае, если браузер не поддерживает такие селекторы, их можно заменить на аналогичные:

input[readonly] {} /\* аналог :read-only \*/

input:not([readonly]) {} /\* аналог :read-write \*/

Однако, обратите внимание, что input:not([readonly]) помимо доступных для редактирования текстовых полей выберет кнопки и другие нетекстовые поля input, например, input[type="submit"].

## Псевдокласс :required [11/18]

Мы уже разбирали, что при помощи специального атрибута required можно отметить поля, обязательные для заполнения

Используя селектор :required можно задать отдельные стили для этих полей

Например

input:required {}

## Псевдокласс :optional [12/18]

Помимо :required существует селектор :optional, выполняющий обратное действие. То есть выберутся все элементы, у которых НЕ указан атрибут required

Пример записи

select:optional {}

## Псевдокласс :checked [13/18]

При помощи селектора :checked можно обратиться ко всем элементам input с типами checkbox или radio, которые являются выбранными (отмеченными)

Например

input:checked {}

Курс «[Селекторы, часть 3](https://htmlacademy.ru/courses/58)»

## Псевдоклассы :invalid и :valid [14/18]

При помощи разных типов полей (email, url и др.) или специфических настроек (pattern, min/max и др.) можно указать браузеру, какие именно данные мы ожидаем от пользователя в том или ином поле.

Селектор :valid выберет все элементы, у которых введенное значение удовлетворяет требованиям.

А селектор :invalid соответственно выберет элементы, у которых введенное значение некорректно.

Пример записи:

input:invalid { }

## Псевдоклассы :in-range и :out-of-range [15/18]

В курсе, посвященном [Формам и HTML5](https://htmlacademy.ru/courses/74) мы разбирали специальный тип поля для ввода числовых значений <input type="number">. У этого поля можно определить максимальное и минимальное значение при помощи атрибутов max и min соответственно.

Селектор :in-range выбирает все элементы, значение которых попадает в указанный диапазон.

А селектор :out-of-range выбирает все элементы, значение которых НЕ попадает в указанный диапазон.

Например:

input:in-range {}

## Объединяй и властвуй [16/18]

Все эти новые селекторы, как и любые другие селекторы, можно комбинировать между собой, соединять с селекторами другого типа, псевдоклассами и т.д. Все зависит только от сложности задачи и необходимости использовать то или иное решение

Например

input[type="checkbox"]:required:checked {}

В данном примере будут выбраны все чекбоксы, которые являются обязательными для заполнения и отмечены галочкой

## Чудеса с ~ и :checked [17/18]

Благодаря селектору :checked, с помощью чистого CSS можно создавать очень много интересных эффектов, так как мы можем не просто выбирать отмеченные поля форм, но и влиять с помощью этих полей на другие элементы.

Для этого нам нужно комбинировать :checked и ~, который позволяет выбрать все элементы, идущие за отмеченным полем. Пример:

input:checked ~ .item {

color: red;

}

Такое CSS-правило задаст красный цвет, всем элементам с классом item, расположенным после отмеченного поля.

Получается, что мы можем с помощью чекбоксов или радиобаттонов управлять внешним видом других элементов. С помощью этого приёма, например, делают переключающиеся вкладки, которые работают без JavaScript.

Давайте разберём пример попроще. Добавим переключатели, которые будут показывать опредёленных котов.

# Рамки и фоны, часть 2

## Размер фона, шаг 1 [2/33]

По умолчанию браузер не меняет размеры фонового изображения. Как быть, если нужно задать фону другие ширину и высоту или привязать его размеры к размерам родительского блока?

Для этого существует свойство background-size. Оно принимает в качестве значения два аргумента: первый — это ширина фонового изображения, второй — его высота. Второй аргумент необязательный. Если высота не указывается, то она определяется автоматически в зависимости от ширины и исходных пропорций. По умолчанию оба аргумента равны auto.

Пример:

background-size: auto auto; /\* исходные ширина и высота изображения \*/

background-size: 100px; /\* ширина 100px, высота пропорциональная \*/

background-size: 100px 200px; /\* ширина 100px, высота 200px \*/

Давайте проверим, как работает это свойство на практике.

## Размер фона, шаг 2 [3/33]

Ещё два значения свойства background-size — это contain и cover.

Значение contain работает так:

* пропорции изображения сохраняются;
* изображению задаются максимально возможные размеры, при которых оно и по ширине, и по высоте полностью помещается в границы фона;
* изображение может не закрывать всю фоновую область блока, если пропорции изображения и блока разные.

Значение cover работает иначе:

* пропорции изображения сохраняются;
* изображению задаются минимально возможные размеры, при которых оно закроет всю фоновую область блока;
* если пропорции изображения и блока разные, то часть изображения обрезается.

Опробуем эти значения на практике.

## Границы фона [4/33]

Ещё одно фоновое свойство — background-origin. Оно задаёт расположение и размеры области отображения фонового изображения и принимает три значения: padding-box, border-box и content-box.



Значение по умолчанию — padding-box. При этом значении область отображения фонового изображения соответствует внутренней области блока, не включая рамки.

При content-box область отображения соответствует только области содержимого, не включая рамки и внутренние отступы.

При border-box область отображения соответствует полному размеру блока, включая и рамки. Фоновое изображение начинается от внешнего края блока и закрывается рамками, если они заданы.

## Обрезка фона [5/33]

Свойство background-clip управляет тем, как обрезается фон. Причём обрезаются не только фоновые изображения, но и фоновый цвет.

Значения свойства такие же, как у background-origin: padding-box, border-box и content-box.



Значение border-box задано по умолчанию, при этом фоновое изображение совсем не обрезается.

Значение padding-box обрежет фон по внутреннему краю области рамки.

Значение content-box обрежет фон по краю области содержимого.

## Множественный фон [6/33]

Можно задать блоку несколько фоновых изображений одновременно. Для этого пути к изображениям в свойстве background-image или background перечисляются через запятую:

background-image: url("image-1.png"), url("image-2.png");

При этом выше будет то изображение, которое находится раньше в списке: image-1.pngв примере будет выше image-2.png.

Значения других свойств для множественных фоновых изображений тоже задаются через запятую. Порядок значений должен соответствовать порядку фоновых картинок. Например:

/\* для первой картинки масштабирование contain, для второй — 100px \*/

background-size: contain, 100px;

/\* для первой картинки позиционирование top, для второй — 100% \*/

background-position: top, 100%;

Давайте Кексу на портрете добавим модные очки с помощью второго фонового изображения.

## Позиция фона от разных сторон [7/33]

Интересная возможность background-position, о которой мы раньше не рассказывали — расположение фона можно задавать относительно любого угла блока, а не только от левого верхнего.

Чтобы указать от какой стороны отсчитывать расположение фона, нужно перед значением координат задать ключевые слова: top, right, bottom или left. Например:

/\* по умолчанию координаты задаются для левого верхнего угла \*/

background-position: 10px 50px; /\* слева 10px, сверху 50px \*/

background-position: right 30px bottom 60px; /\* справа 30px, снизу 60px \*/

background-position: left 50px bottom 10px; /\* слева 50px, снизу 10px \*/

background-position: right 40px top 30px; /\* справа 40px, сверху 30px \*/

Поддержка данных значений свойства background-position в современных браузерах [практически полная](http://caniuse.com/#feat=css-background-offsets).

Давайте добавим мсьё Кексу модные монокль и усы, а расположим эту картинку от нижнего правого угла.

Курс «[Рамки и фоны, часть 2](https://htmlacademy.ru/courses/88)»

## Повторение фона: background-repeat: round [8/33]

Два интересных, но малоизвестных значения привычного свойства background-repeat, которое задаёт повторение фона, — это round и space.

Значение свойства по умолчанию background-repeat: repeat просто повторяет фоновую картинку по всей ширине и высоте блока. Если части повторяющейся картинки не помещаются в ширину блока, то они просто обрезаются.

Если задать значение background-repeat: round, то повторяющиеся картинки по краям блока обрезаться не будут, а равномерно растянутся или сожмутся по всей ширине, чтобы занять оставшееся пространство.

Кстати, background-repeat принимает в качестве значения два аргумента: режим повторения по горизонтали и по вертикали. Если передать один параметр, то он применится к обоим направлениям. Например:

/\* повторение фона repeat по горизонтали и вертикали \*/

background-repeat: repeat;

/\* повторение фона round по горизонтали и repeat по вертикали \*/

background-repeat: round repeat;

Давайте опробуем режим повторения round на практике.

## Повторение фона: background-repeat: space [9/33]

Режим повторения фона background-repeat: space похож на round, при нём части картинки тоже не обрезаются. Отличие в том, что повторяющиеся фоновые картинки не сжимаются или растягиваются, а для компенсации ширины блока между ними добавляется пустое пространство.

Попробуем, как работает это свойство.

## Внешняя рамка [10/33]

Полезное, но редко используемое свойство — внешняя рамка или outline.

Синтаксис outline схож с border. Но внешней рамке нельзя задать параметры отдельных сторон. Пример:

/\* сплошная чёрная рамка толщиной 10px \*/

outline: 10px solid black;

Это сокращённый синтаксис свойства outline. Есть и отдельные свойства:

outline-width: 10px; /\* толщина \*/

outline-style: solid; /\* стиль \*/

outline-color: black; /\* цвет \*/

Внешняя рамка всегда отображается снаружи элемента, не влияет на его размер и не занимает места, то есть отображается над другими элементами.

C помощью свойства outline-offset можно изменять расположение рамки. Положительное значение отодвигает рамку от внешнего края элемента, а отрицательное «втягивает» внутрь.

У outline-style те же значения, что и у border-style. Кстати, есть интересные типы рамок ridge и groove, они создают объёмные рамки.

Попробуем всё это на практике.

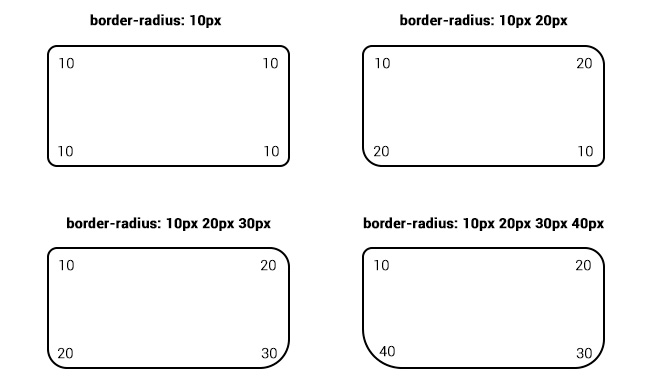
## Скругление углов, часть 1 [11/33]

Закруглять углы элементов можно с помощью свойства border-radius.

Свойство задаёт радиус скругления углов в пикселях или процентах. Если у блока есть рамка, то скругляется и она.



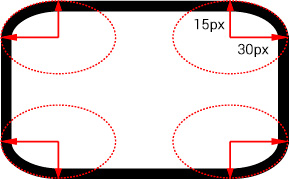
Свойство border-radius принимает от одного до четырёх аргументов.



## Скругление углов, часть 2 [12/33]

Вы можете закруглять отдельные углы с помощью свойств: border-top-left-radius,border-top-right-radius, border-bottom-right-radius и border-bottom-left-radius.

А ещё можно задавать разные горизонтальные и вертикальные радиусы скругления. Для этого нужно написать два значения через пробел в свойстве скругления угла. Первое значение задаёт радиус по горизонтали, второе — по вертикали:



/\* горизонтальный радиус 30px, вертикальный 15px \*/

border-top-right-radius: 30px 15px;

Разные горизонтальные и вертикальные радиусы можно задавать и в свойстве border-radius. Для этого нужно использовать /, например:

/\* горизонтальный радиус всех углов 10px, вертикальный 5px \*/

border-radius: 10px / 5px;

/\* разные горизонтальные и вертикальные радиусы у каждого угла \*/

border-radius: 10px 20px 30px 40px / 5px 15px 25px 35px;

## Изображение рамки: border-image-source [13/33]

Мы подошли к обширной и интересной теме, в которой рамки и фоновые изображения встречаются.

Итак, семейство свойств border-image задаёт фоновое изображение для рамки блока. Поддержка данного семейства свойств в современных браузерах [довольно неплохая](http://caniuse.com/#feat=border-image).

Свойство border-image-source задаёт путь к изображению рамки. По умолчанию картинкой заполнятся только углы рамки. В следующих заданиях мы разберём, как можно управлять отображением рамки.

В качестве изображения для рамки используем вот такую картинку:



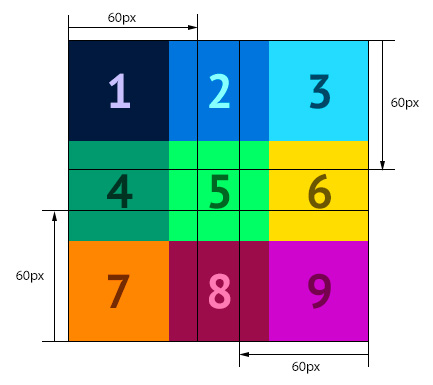
Синтаксис свойства такой же, как у background-image, то есть:

border-image-source: url("image.jpg");

Давайте же зададим фоновое изображение рамки и начнём его настраивать.

## Изображение рамки: border-image-slice [14/33]

Давайте разберёмся, как работает механизм «нарезки» фонового изображения для рамки.



Каждая рамка имеет 9 областей: 4 угла, 4 стороны и центральную область. Для заполнения этих областей браузер должен нарезать картинку для рамки на 9 частей. Когда браузер не знает, как это сделать, он просто размещает картинку по углам — мы видели это в предыдущем задании.

Свойство border-image-slice задаёт отступы от краёв картинки до четырёх линий, которые «разрезают» её на части, как на схеме справа. Если эти отступы небольшие, то получается «нарезка» из 9частей, которые затем размещаются в соответствующих областях рамки.

Но если отступы слишком большие (больше половины картинки), то браузер не может получить 9 частей и располагает то, что отрезалось по углам.

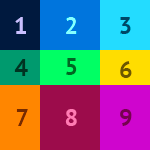
Значение свойства можно задавать числом без единицы измерения (оно обычно обозначает пиксели) или в процентах (относительно размера самой картинки). Пример:

border-image-slice: 60;

border-image-slice: 10%;

## Нарезка несимметричных картинок [15/33]

Сравните две картинки:

Для нарезки первой из них можно было задать одинаковые отступы линий разреза — 50px. Для второй картинки этого явно недостаточно — она менее симметрична.

С помощью border-image-slice можно задавать разные отступы линий разреза. Для этого нужно задавать значения через пробел в порядке: верх, право, низ, лево. Пример:

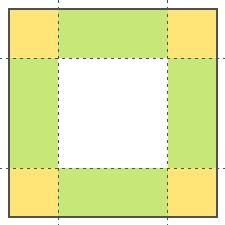
border-image-slice: 10 20 30 40;

Средняя часть картинки обычно не используется. Но если в значение свойства добавить ключевое слово fill, то средняя часть картинки будет отображаться в средней области рамки: она закроет собой фон блока, но не закроет содержимое. Пример:

border-image-slice: 10 20 30 40 fill;

Курс «[Рамки и фоны, часть 2](https://htmlacademy.ru/courses/88)»

## Изображение рамки: border-image-repeat, часть 1 [16/33]



Свойство border-image-repeat задаёт способ заполнения фоном боковых сторон рамки (зелёные области на рисунке).

У свойства четыре значения: stretch, repeat, space и round.

Значение по умолчанию — stretch. При этом значении фоновые картинки растягиваются на всю длину боковых сторон.

Если задано значение repeat, то фоновые картинки будут повторяться. При этом они могут обрезаться.

Можно устанавливать режим заполнения отдельно для горизонтальных и вертикальных сторон рамки.

Например:

/\* все стороны рамки заполняются в режиме stretch \*/

border-image-repeat: stretch;

/\* горизонтальные стороны — режим repeat, вертикальные — stretch \*/

border-image-repeat: repeat stretch;

Давайте посмотрим, в чём различие этих двух вариантов.

## Изображение рамки: border-image-repeat, часть 2 [17/33]

Значение round свойства border-image-repeat тоже устанавливает режим заполнения стороны рамки повторяющимися боковыми участками картинки. Но, в отличие от repeat, если в ширину стороны не вмещается целое число повторящихся кусочков, крайние части не обрезаются. Кусочки при этом равномерно растягиваются так, чтобы все они стали одного размера и заняли оставшееся место стороны рамки.

Согласно спецификации при заданном значении space алгоритм похож на round, только для компенсации оставшегося места кусочки картинки не растягиваются, а остаются прежней ширины, при этом между кусочками появляется дополнительное свободное пространство.

На момент создания курса в современных браузерах реализация свойства spaceидентична свойству repeat.

Проверим значения свойств на практике.

## Изображение рамки: border-image-width [18/33]

Следующее свойство, которое мы рассмотрим — border-image-width.

У блока должна существовать рамка определённой толщины border-width, тогда ему можно задать и фоновую картинку для рамки. Область, в которой будет отображаться эта картинка по умолчанию равна ширине рамки.

Свойство border-image-width позволяет управлять шириной видимой области рамки-картинки, масштабировать её. Саму ширину рамки это свойство не меняет.

Если значение этого свойства больше border-width, картинка рамки заползёт под содержимое, даже если не задано свойство fill.

Ширина рамки-картинки задаётся в %, px, em или других единицах измерения. Также возможно значение auto, при котором ширина зависит от значения border-image-slice.

Можно задавать разную ширину сторон. В этом случае значения перечисляются аналогично margin, padding в последовательности: верхнее, правое, нижнее, левое. Например:

border-image-width: 10px 20px 30px 40px;

border-image-width: 10px 50px;

Попробуем поуправлять шириной рамки-картинки.

## Изображение рамки: border-image-outset [19/33]

Ещё одно свойство, относящееся к фоновому изображению рамки, border-image-outset. Аналогично outline-offset это свойство позволяет отодвинуть рамку за пределы элемента, но при этом одновременно немного масштабируя картинку. Отрицательные значения border-image-outset не поддерживаются.

Отступы рамок-изображений тоже можно задавать разные для каждой из сторон. Синтаксис обычный:

border-image-outset: 10px;

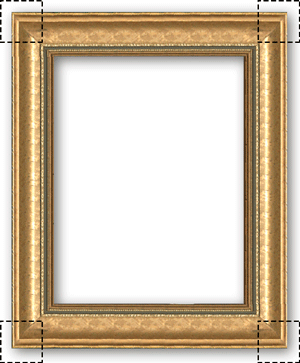
border-image-outset: 10px 20px 30px 40px;

Давайте немного отодвинем рамку у портрета.

## Королевская рамка Кексика [20/33]

Теперь давайте попрактикуемся в изготовлении рамок «на кошках». Точнее «на коте» и разных рамках к его портретам.

Первой рамкой возьмём изображение настоящей рамы картины. Для уголков рамки возьмём небольшие области в углах картинки:



Также попробуем «разрезать» исходную картинку с заполнением пространства между сторонами рамки. Так как у картинки посередине прозрачная область, то сквозь неё будет видно Кекса.

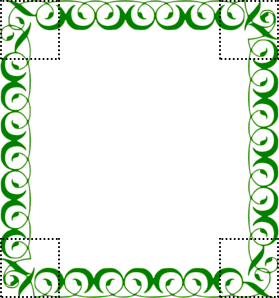
И ещё в конце зададим графической рамке небольшой внутренний отступ.

1. **Цель 1**Портрету задайте фоновое изображение рамки img/frame.png,
2. **Цель 2**свойство border-image-slice со значением 100,
3. **Цель 3**а затем со значением 100 fill,
4. **Цель 4**а потом задайте border-image-outset со зачением 10px.

## Эко-рамка Кексика [21/33]

У прошлой графической рамки был задан режим повторения stretch, который задаётся по умолчанию. На сторонах рамки не было видно сильных искажений при растяжении исходной картинки, так как она имела большие размеры сама по себе.

Картинка для графической рамки в этом задании небольшая и имеет более неравномерный узор, чем золотая рамка предыдущего задания. Давайте возьмём следующие её части для заполнения рамки:



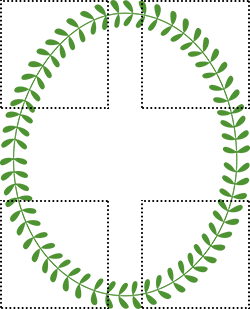
С такой картинкой будут хорошо видны режимы повторения графической рамки: при растяжении стороны картинки существенно искажаются, поэтому мы попробуем разные значения повторения, чтобы найти оптимальный.

1. **Цель 1**Портрету задайте фоновое изображение рамкиimg/eco-frame.png и свойство border-image-slice со значением 60,
2. **Цель 2**затем свойство border-image-repeat со значением repeat
3. **Цель 3**и со значением round.

## Круглая рамка Кексика, часть 1 [22/33]

К сожалению, свойство border-radius не влияет на border-image, так что единственный способ сделать округлую рамку — использовать круглую картинку.

В этом и следующем задании мы сделаем круглый портрет Кекса в круглой рамке. Для этого нарежем картинку для рамки следующим образом:



Также мы используем режим fill, так как картинка для рамки прозрачная внутри.

1. **Цель 1**Портрету задайте фоновое изображение рамкиimg/leafs-frame.png,
2. **Цель 2**свойство border-image-slice со значением 180
3. **Цель 3**затем добавьте fill в border-image-slice,

## Круглая рамка Кексика, часть 2 [23/33]

Теперь, когда графическая рамка готова, закруглим сам портрет Кекса, затем скроем лишний фон с помощью обрезки background-clip и уберём вспомогательную внешнюю рамку.

И круглый портрет в круглой рамке готов!

## Рамки и треугольники, часть 1 [25/33]

В этой серии заданий мы рассмотрим интересный приём, связанный со старым добрым свойством border. Вы можете использовать обычные рамки для создания необычных эффектов!

У рамок есть одна особенность: если рамка элемента широкая, а сам элемент имеет нулевую ширину и высоту, то стороны рамки становятся треугольными.

Убедимся на практике!

## Рамки и треугольники, часть 2 [26/33]

Теперь чтобы сделать треугольник нужно:

* одной стороне рамки задать нулевую ширину,
* ещё две стороны сделать прозрачными,
* последней стороне задать нужный цвет.

Так с помощью обычного border можно создавать треугольные стрелки, направленные во все стороны.

Давайте создадим стрелку-треугольник, направленную вниз.

Измените рамку у блока .arrow-bottom:

1. **Цель 1**задайте цвет верхней стороны #0074d9,
2. **Цель 2**толщину нижней стороны 0,
3. **Цель 3**цвет левой и правой сторон сделайте прозрачными transparent.

## Рамки и треугольники, часть 3 [27/33]

Похожим способом можно создавать треугольные стрелки, направленные по диагонали.

Для этого:

* одной стороне рамки задаём нужный цвет,
* ещё двум сторонам задаём нулевую ширину,
* последнюю сторону делаем прозрачной.

Давайте создадим стрелку-треугольник, направленную по диагонали вверх и вправо.

Измените рамку у блока .arrow-top-right:

1. **Цель 1**задайте цвет верхней стороны #0074d9,
2. **Цель 2**толщину нижней стороны 0,
3. **Цель 3**затем толщину правой стороны 0
4. **Цель 4**и цвет левой стороны transparent.
5. Курс «[Рамки и фоны, часть 2](https://htmlacademy.ru/courses/88)»

## Стрелка с помощью рамки [28/33]

1. А теперь давайте добавим стрелке «хвост».
2. Принцип построения такой же, как и раньше, но теперь треугольником будет не сам элемент, а его псевдоэлемент. Сам же элемент будет «хвостом» стрелки.
3. Стрелка будет указывать вправо.
4. **Цель 1**Псевдоэлементу .arrow-right-long::after задайте сплошную рамку толщиной 40px и цветом #ffffff,
5. **Цель 2**затем задайте ему позицию top: -25px и right: -30px,
6. **Цель 3**обнулите толщину правой стороны рамки, а левой задайте цвет #0074d9,
7. **Цель 4**а затем верхнюю и нижнюю стороны рамки сделайте прозрачными.

.arrow-right-long {

position: relative;

width: 200px;

height: 30px;

margin: 150px auto;

background: #0074d9;

}

.arrow-right-long::after {

content: "";

position: absolute;

border:40px solid #ffffff;

top:-25px;

right:-30px;

border-right:0;

border-left:40px solid #0074d9;

border-bottom:40px solid transparent;

border-top:40px solid transparent;

}

## Круглая стрелка с помощью рамки, часть 1 [29/33]

Следующим шагом давайте применим изученный приём к круглому блоку.

В результате получится стрелка со скруглённым «хвостом».

Для начала сделаем круглое основание.

1. **Цель 1**Блоку .arrow-round задайте сплошную рамку толщиной 50px и цветом #0074d9,
2. **Цель 2**затем задайте скругление углов рамки со значением 50%,
3. **Цель 3**а правой части рамки задайте прозрачный цвет transparent

.arrow-round {

position: relative;

width: 200px;

height: 200px;

margin: 100px auto;

border:solid 50px #0074d9;

border-radius:50%;

border-right:50px transparent solid;

## Круглая стрелка с помощью рамки, часть 2 [30/33]

Теперь займёмся самим указателем. Сделаем его по аналогии с прошлыми заданиями.

В итоге получим простую в исполнении круглую CSS-стрелку, которую спокойно можно использовать в интерфейсах.

Псевдоэлементу .arrow-round::after:

1. **Цель 1**Задайте сплошную рамку толщиной 100px и цветом #ffffff, цвет нижней стороны рамки — #0074d9.
2. **Цель 2**Задайте координаты top: -45px и left: 130px.
3. **Цель 3**Обнулите толщину правой и верхней сторон рамки,
4. **Цель 4**левую сторону рамки сделайте прозрачной.

.arrow-round {

position: relative;

width: 200px;

height: 200px;

margin: 100px auto;

border: 50px solid #0074d9;

border-radius: 50%;

border-right-color: transparent;

}

.arrow-round::after {

content: "";

position: absolute;

border:solid 100px #ffffff;

border-bottom:#0074d9 solid 100px;

top:-45px;

left:130px;

border-right:0;

border-top:0;

border-left:solid 100px transparent;

}

## Треугольники в жизни, часть 1 [31/33]

А сейчас давайте посмотрим, как изученные ранее треугольники используются в реальных задачах.

Самый очевидный пример — иконки сортировки в каталогах товаров: треугольники, направленные вверх и вниз.

Почему их выгодно делать с помощью рамок? Потому что CSS-стрелкам можно легко менять пропорции, задавать разные цвета и размеры, а это очень гибко.

Обычно управляющие элементы для сортировки размечают с помощью ссылок. Мы поступим так же: внутри ссылки будет поясняющий текст, а в стрелку превратим псевдоэлемент ссылки.

## Треугольники в жизни, часть 2 [32/33]

Теперь добавим дополнительные классы arrow-up и arrow-down, которые будут обозначать направление стрелки вверх и вниз.

Для стрелки вверх нужно будет убрать верхнюю рамку у псевдоэлемента, а у стрелки вниз — нижнюю рамку. Чтобы стрелки вели себя в потоке, как текст, зададим псевдоэлементам блочно-строчный тип.

И последним шагом скроем текст подписи у ссылки, для этого просто зададим нулевой размер шрифта.

Есть и более сложный вариант стрелок для сортировки: когда у каждого параметра создаются сразу две стрелки. Этот вариант делается схожим образом, но для создания стрелок используются оба псевдоэлемента: и ::before, и ::after.

1. **Цель 1**Для .arrow-down::before обнулите толщину нижней рамки,
2. **Цель 2**затем для .arrow-up::before обнулите толщину верхней рамки.
3. **Цель 3**Для .arrow::before задайте свойство display: inline-block,
4. **Цель 4**а затем самим ссылкам .arrow обнулите размер шрифта.

div class="goods">

<p>

Сортировка:

<a class="arrow arrow-up" href="#">по возрастанию</a>

<a class="arrow arrow-down" href="#">по убыванию</a>

a.arrow {

display: inline-block;

margin: 0 10px;

font-size:0;

}

.arrow::before {

content: "";

border: 20px solid #0074d9;

border-right-width: 10px;

border-left-width: 10px;

border-right-color: transparent;

border-left-color: transparent;

display:inline-block;

}

.arrow-up::before {

border-top:0;

}

.arrow-down::before {

border-bottom:0;

}

# Игра теней

## Свойство box-shadow

Тени создаются с помощью CSS-свойства box-shadow. Это достаточно сложное свойство, которое состоит из 6 компонентов. В этом курсе мы детально рассмотрим, как оно работает, а также научимся некоторым интересным приёмам.

С помощью этого свойства можно создавать множественные тени. Мы тоже потренируемся это делать.

А пока посмотрим на пример работы этого свойства.

## Смещение тени по горизонтали

Мы создали простую тень, теперь разберемся с синтаксисом свойства. Вот он:

box-shadow:

[inset] — внутренняя

5px <— смещение по x

10px — смещение по y

[2px] — размытие

[3px] — растяжение

[red]; — цвет

Ключевое слово inset, растяжение, размытие и цвет не являются обязательными. Порядок следования цифровых значений важен. Первое цифровое значение означает смещение по горизонтали.

Смещение по горизонтали может быть положительным и отрицательным.

## Смещение тени по вертикали

Еще раз синтаксис:

box-shadow:

[inset] — внутренняя

5px — смещение по x

10px <— смещение по y

[2px] — размытие

[3px] — растяжение

[red]; — цвет

Второе цифровое значение задаёт смещение по вертикали. Оно работает аналогично предыдущему значению, смещая тень вверх или вниз.

## Размытие тени

Третье цифровое значение задаёт радиус размытия тени.

box-shadow:

[inset] — внутренняя

5px — смещение по x

10px — смещение по y

[2px] <— размытие

[3px] — растяжение

[red]; — цвет

Чем меньше значение, тем четче тень. Максимальная четкость достигается при нулевом значении. Размытие не может быть отрицательным. Если размытие не указано, то оно равно нулю.

## Растяжение тени

Четвёртое цифровое значение задаёт растяжение тени.

box-shadow:

[inset] — внутренняя

5px — смещение по x

10px — смещение по y

[2px] — размытие

[3px] <— растяжение

[red]; — цвет

По умолчанию размер и форма тени такие же, как и у элемента, которому она принадлежит. Но размер тени можно изменить с помощью растяжения.

При нулевом (или не указанном) растяжении размер тени такой же, как у элемента. При положительном растяжении размер тени становится больше размера элемента, а при отрицательном — меньше.

## Размытие + растяжение

Размытие и растяжение иногда используют друг с другом, это позволяет управлять насыщенностью тени. Внешний вид тени будет зависеть от соотношения величин размытия и растяжения:

1. размытие есть, растяжение отрицательное — тень очень светлая;
2. размытие есть, растяжения нет — тень обычная;
3. размытие больше растяжения — тень темнее обычного;
4. размытие меньше растяжения — тень очень темная.

Впрочем, поэкспериментируйте самостоятельно.

.btn1 {

box-shadow: 0 0 6px -2px #34495e;

}

.btn2 {

box-shadow: 0 0 6px 0 #34495e;

}

.btn3 {

box-shadow: 0 0 6px 3px #34495e;

}

.btn4 {

box-shadow: 0 0 3px 6px #34495e;

}

## Цвет тени

Цвет тени задается с помощью любого цветового формата CSS. Если цвет тени не задан, то он берётся от цвета текста элемента с тенью. Именно поэтому в текущем задании исходный цвет тени — жёлтый.

box-shadow:

[inset] — внутренняя

5px — смещение по x

10px — смещение по y

[2px] — размытие

[3px] — растяжение

[red]; <— цвет

Цвет тени — мощный инструмент в руках умелого дизайнера, который позволяет создавать выразительные эффекты.

## Внутренняя тень

Если в box-shadow использовать ключевое слово inset, то тень будет отображаться внутри элемента.

box-shadow:

[inset] <— внутренняя

5px — смещение по x

10px — смещение по y

[2px] — размытие

[3px] — растяжение

[red]; — цвет

inset лучше размещать либо в самом начале свойства, либо в самом конце.

## Полупрозрачная тень

Если у тени ненулевой радиус размытия, то она автоматически становится полупрозрачной. Но иногда нужно сделать полупрозрачной тень без радиуса размытия (например, как у кнопок в мини-браузере).

Для этого можно использовать новый формат записи цветов в CSS — rgba. Пример:

color: red;

color: #ff0000;

color: rgba(255, 0, 0, 0.5);

Все записи цвета в примере задают красный цвет, но в последней записи задан красный с 50% прозрачностью.

В rgba первые три значения задают цвет и могут быть целым числом от 0 до 255. Последнее значение задаёт уровень прозрачности и может быть дробным числом от 0 (полная прозрачность) до 1.

Задавать полупрозрачный цвет с помощью rgba можно не только теням, но и остальным свойствам с цветом: рамкам, фону и так далее.

## Тень только с одной стороны

Теперь рассмотрим несколько интересных приёмов работы с тенями.

Первый приём позволяет сделать так, чтобы тень была видна только с одной стороны элемента.

Для этого нужно задать тени отрицательное растяжение и положительное размытие, а затем сместить тень в нужном направлении. Отрицательное растяжение сделает размытую тень меньше элемента, а смещение покажет край тени только с одной стороны.

Подобный приём позволяет имитировать градиенты с помощью внутренних теней.

## Множественные тени

Свойство box-shadow позволяет задавать одному элементу несколько теней. Для этого нужно перечислить значения теней через запятую, например:

box-shadow:

0 0 5px 0 red,

0 0 10px 0 white;

Тени, которые расположены в списке выше, будут перекрывать тени, которые расположены ниже.

box-shadow:

0 -5px 10px -3px #3498db,

0 5px 10px -3px #2ecc71;

## Несколько рамок у одного элемента

С помощью множественных теней можно создавать различные интересные эффекты.

Например, можно имитировать множественные рамки у элемента. Для этого нужно добавить элементу несколько теней с положительным растяжением и с нулевыми смещениями и размытием.

Но тут есть одна тонкость. Настоящие рамки увеличивают размер элемента, а рамки-тени — нет. Поэтому надо следить, чтобы вокруг элемента было достаточно свободного пространства, иначе он может перекрыть соседние элементы.

## Теневое искусство – 1

Самое интересное применение теней — это рисование. Умельцы создают сложнейшие изображения с помощью теней. Мы тоже создадим не очень сложную, но узнаваемую картинку на чистом CSS.

В исходном коде уже есть заготовка — скруглённый элемент с размерами 100 на 100 пикселей без рамок и фона. Тени имеют тот же размер и форму, что и их родительский элемент, поэтому добавляемые нами тени будут круглыми.

Перед началом пара замечаний:

1. Если какой-то параметр тени не указан, значит он равен 0.
2. Необязательно задавать все варианты свойства box-shadow (префиксованные и нет). Используйте тот вариант, который работает в вашем браузере (серверная проверка использует вариант без префикса).

body {

padding: 20px;

background-color: white;

}

.circle {

margin: 10px auto;

width: 100px;

height: 100px;

border-radius: 50%;

box-shadow:

-5px -5px 5px 3px white inset,

0 0 0 9px #6c6d70 inset,

0 0 0 10px black inset;

}

## Теневое искусство – 2

Итак, мы создали заготовку круга, используя внутренние тени.

На этом шаге завершим заготовку, добавив еще несколько теней. Кстати, тоже внутренних.

Принцип построения простой. Мы каждый раз добавляем всё более узкую тень в начало списка теней и она частично перекрывает нижележащие тени. Внутренние тени выбраны потому, что они не выходят за границы элемента.

body {

padding: 20px;

background-color: white;

}

.circle {

margin: 10px auto;

width: 100px;

height: 100px;

border-radius: 50%;

box-shadow:

inset 2px 2px 2px 0 white,

inset 0 0 0 4px #4f5054,

inset 0 0 0 5px black,

inset -5px -5px 5px 3px white,

inset 0 0 0 9px #6c6d70,

inset 0 0 0 10px black;

}

## Теневое искусство – 3

Подготовка к созданию картины завершена.

У нас есть класс, который создает красивое кольцо с помощью теней. В исходном коде также добавлены элементы и стили, чтобы расположить кольца так, как нам надо.

Вам остаётся только добавить класс к четырем элементам и получится картинка. Напомним, что несколько классов задаются элементу через пробел:

<div class="one two">

<!DOCTYPE html>

<html lang="ru">

<head>

<meta charset="utf-8">

<title>Теневое искусство – 3</title>

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

</head>

<body>

<div class="logo">

<div class="c1 circle"></div>

<div class="c2 circle"></div>

<div class="c3 circle"></div>

<div class="c4 circle"></div>

</div>

</body>

</html>

body {

padding: 20px;

background-color: white;

}

.logo {

margin: 50px auto;

width: 280px;

}

.logo div {

position: absolute;

}

.logo .c1 {

margin-left: 0;

}

.logo .c2 {

margin-left: 60px;

}

.logo .c3 {

margin-left: 120px;

}

.logo .c4 {

margin-left: 180px;

}

.circle {

margin: 10px auto;

width: 100px;

height: 100px;

border-radius: 50%;

box-shadow:

inset 2px 2px 2px 0 white,

inset 0 0 0 4px #4f5054,

inset 0 0 0 5px black,

inset -5px -5px 5px 3px white,

inset 0 0 0 9px #6c6d70,

inset 0 0 0 10px black;

}

# [Линейные градиенты](https://htmlacademy.ru/courses/70)

## Hello, linear-gradient! [1/23]

В математике градиент — это функция, показывающая направление наискорейшего возрастания некоторой величины, значение которой меняется от одной точки пространства к другой.

Если применить градиент к цвету, то получится плавный переход от одного цвета к другому. Например, вот такой:

Градиенты описываются внутри CSS-свойства background-image. Простейший градиент можно описать таким образом:

background-image: linear-gradient(yellow, green);

Сам градиент создаётся функцией linear-gradient, в параметрах которой указываются направление градиента и набор цветов. Направление можно не указывать, тогда будет использовано значение по умолчанию — сверху вниз. Цветов может быть любое количество.

Давайте потренируемся и создадим простейший двухцветный градиент для блока .content.

Спецификация градиентов только недавно устоялась. В этом курсе мы используем новейший синтаксис градиентов. Поэтому лучше обновить браузер.

Если у вас старый браузер, то лучше проверять код с помощью серверной проверки. Также могут возникнуть проблемы с отображением градиентов в мини-браузере.

## Направление градиента [2/23]

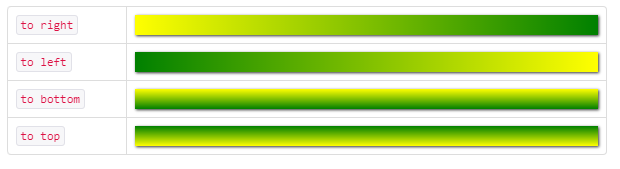
Направления градиента задаются с помощью ключевых слов: top, bottom, left, right.

Направление градиента располагается перед списком цветов и включает в себя частицу to. Она была добавлена в синтаксис для улучшения читабельности и наглядности:

background-image: linear-gradient(to right, yellow, green);

И сразу понятно, что это: «Жёлто-зелёный градиент слева направо».

Вот примеры разных направлений градиента c цветами yellow, green:



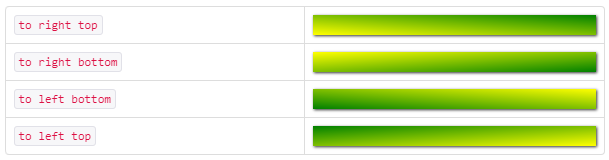
А теперь самостоятельно потренируйтесь задавать разные направления градиентов.

## Градиенты по диагоналям [3/23]

Градиенты можно направлять по диагонали, из угла в угол. Для этого нужно комбинировать top, bottom и left, right. Например, градиент, идущий из левого нижнего в правый верхний угол:

background-image: linear-gradient(to right top, yellow, green);

Вот примеры диагональных градиентов c цветами yellow, green:



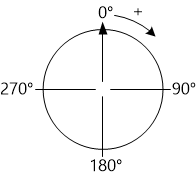
## Градиенты под углом [4/23]

Направление линейного градиента можно задавать и в виде произвольного угла, например, 245°. Направление в градусах задаётся с помощью единицы измерения deg. Можно задавать положительные и отрицательные углы. Примеры:

background-image: linear-gradient(90deg, yellow, green);

background-image: linear-gradient(-135deg, yellow, green);

Углы градиента отсчитываются так:



0deg соответствует 12 часам, а отсчёт угла идет по часовой стрелке. В случае, если угол задан отрицательным значением, например, -90deg, то он отсчитывается против часовой стрелки.

В этом задании нужно прописать оставшимся трём «листам» градиенты по аналогии с первым: от цветного края «листа» к чёрному центру «цветка».

## Диагонали против градусов [5/23]

Иногда градиенты с направлениями, заданными с помощью градусов и диагоналей, выглядят одинаково.

|  |  |
| --- | --- |
| to right top | 45deg |

Однако их поведение отличается. Градиенты, заданные с помощью градусов, не зависят от формы контейнера, а диагональные градиенты зависят. Диагональные градиенты всегда остаются привязанными к своим углам.

Конечно, если контейнеры квадратные, то отличий не видно. Но что будет, если вытянуть их по высоте? Давайте проверим.

## Nyan cat наносит ответный удар [7/23]

В линейный градиент можно включать больше двух цветов. Для этого цвета просто перечисляются через запятую. Например, если задать такой CSS:

linear-gradient(to right, red, yellow, green)

То получится равномерный градиент c тремя цветами:

В этом задании нужно блоку .rainbow прописать радужный градиент в направлении сверху вниз и заставить Nyan cat лететь!

## Пропорции цветов и колорстопы [8/23]

По умолчанию цвета в градиентах распределяются равномерно, в одинаковых пропорциях, но этим поведением можно управлять.

Делается это с помощью так называемых колорстопов, которые записываются сразу после значений цветов, например, red 0%, yellow 100%.

Колорстоп указывает положение цвета в градиенте, его можно задавать в процентах, пикселях и других единицах. Давайте рассмотрим несколько примеров, чтобы понять поведение колорстопов:

Колорстоп задаёт то место, где будет располагаться центральная (самая насыщенная) часть цвета.

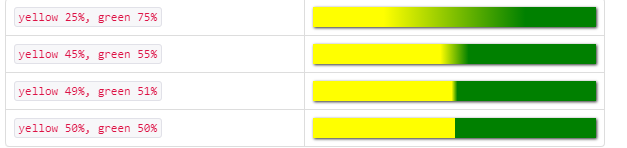


## Резкие переходы цветов [9/23]

Позиция цвета (или колорстоп) задаёт расположение центральной части цвета, ту точку, от которой начинается переход в другой цвет.

А что будет, если задать для соседних цветов одну и ту же позицию? В этом случае получится резкий переход цветов, так как они оба будут «вытекать» из одной точки в противоположных направлениях.

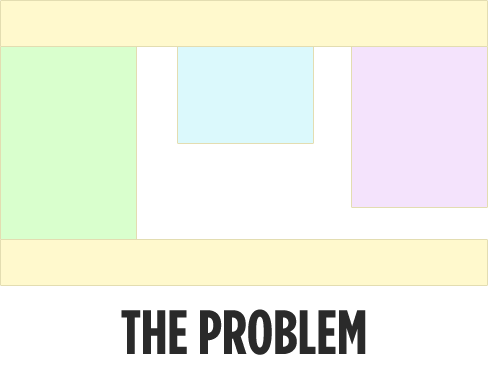
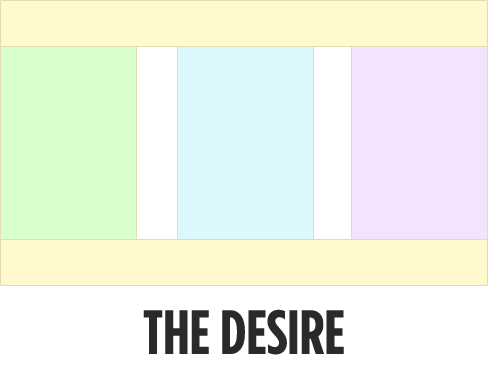
Легче продемонстрировать это поведение на примере:



Этот приём часто используют для создания интересных эффектов.

## Псевдоколонки на градиентах [10/23]

Одна из интересных задач вёрстки — как сверстать тянущиеся колонки одинаковой высоты без использования таблиц? Проблема эта связана с особенностями поведения блочной модели и проиллюстрирована на картинках ниже:

Для решения этой задачи существует несколько приёмов, которые описаны в [этой статье на английском](http://css-tricks.com/fluid-width-equal-height-columns). Кстати, картинки тоже взяты оттуда.

Давайте применим приём с использованием градиентов с резкими переходами, которые имитируют тянущиеся колонки. Фон с такими градиентами обычно размещают в контейнере колонок. Пропорции градиентов подбирают такими же, как и пропорции колонок, чтобы они тянулись одинаково.

## Цветовые переходы в px [12/23]

Цветовые переходы градиента можно также задавать в пикселях. Работают они по аналогии с процентными переходами. Отличие заключается в том, что процентные переходы зависят от размера элемента, а переходы в пикселях — нет.

В этом задании нужно дополнить картинку, прописав градиенты оставшихся листов.

## Полупрозрачные градиенты [13/23]

В CSS вы можете задать любой цвет с полупрозрачностью или даже задать прозрачный цвет. Это делается с помощью rgba-значений цветов, в которых помимо трёх цветовых компонент присутствует прозрачность. Примеры:

* rgba(255, 255, 255, 1) — обычный белый цвет.
* rgba(255, 255, 255, 0.5) — наполовину прозрачный белый.
* rgba(255, 255, 255, 0) — полностью прозрачный цвет.

Также прозрачный цвет можно задать с помощью ключевого слова transparent.

Прозрачные и полупрозрачные цвета используют и в градиентах для создания интересных эффектов.

В этом задании нужно дополнить картинку, прописав полосатые полупрозрачные градиенты двух оставшихся квадратов по аналогии с двумя имеющимися, чтобы в итоге в центре возник ещё один квадрат.

## Повторяющийся линейный градиент [14/23]

Помимо обычных градиентов существуют и повторяющиеся. Их синтаксис полностью совпадает с синтаксисом обычных, только вместо linear-gradient пишется repeating-linear-gradient. Повторяющийся градиент хорош для создания полосатых фонов или фонов-орнаментов средствами CSS.

Есть несколько тонкостей, которые нужно знать про повторяющиеся градиенты:

1. Размер фрагмента определяется по последнему колорстопу. Чтобы повторение было видно, последний колорстоп должен быть меньше, чем размер элемента с градиентом.
2. Если первый и последний цвета градиента различаются, то будут видны резкие границы между повторяющимися фрагментами. Чтобы от них избавиться, нужно задавать одинаковый первый и последний цвета.
3. Колорстопы в повторяющихся градиентах обычно задают в пикселях, но можно использовать и проценты.

Кстати, вместо повторяющихся градиентов можно использовать обычные градиенты в сочетании со свойствами background-size и background-repeat. Но повторяющиеся градиенты удобнее и требуют меньше кода.

В этом задании мы будем делать из обычных градиентов повторяющиеся. Обратите внимание на то, какого размера получаются фрагменты. Например, в первых двух блоках поместится ровно четыре фрагмента.

## Строим сложный фон — часть 1 [15/23]

В этой серии заданий мы построим сложный узор с помощью градиентов. Делать это мы будем с помощью техники множественных фоновых изображений.

В CSS можно задавать элементу сразу несколько фоновых картинок, перечисляя их через запятую:

background-image:

url('img1.png'),

url('img2.png'),

url('img3.png');

Точно так же вместо картинок можно использовать градиенты:

background-image:

linear-gradient(...),

linear-gradient(...),

linear-gradient(...);

Мы будем использовать градиенты с прозрачными цветами и резкими переходами. Сначала мы построим повторяющийся фрагмент, а затем размножим его. В этом задании вы добавите первые два градиента и получиться у вас должно следующее:



background-image:

linear-gradient(135deg,white 25%, transparent 25%),

linear-gradient(225deg,white 25%, transparent 25%);

## Строим сложный фон — часть 2 [16/23]

Продолжим делать узор.

На этом шаге вам нужно добавить еще два градиента, чтобы получился фрагмент в виде чёрного ромба.

background-image:

linear-gradient(135deg, white 25%, transparent 25%),

linear-gradient(225deg, white 25%, transparent 25%),

linear-gradient(45deg, white 25%, transparent 25%),

linear-gradient(315deg, white 25%, transparent 25%);

## Строим сложный фон — часть 3 [17/23]

CSS-градиенты — это особые фоновые изображения, и на них действуют все свойства для управления фонами: background-position, background-size, background-repeat.

В отличие от обычных изображений градиенты не имеют «собственного» размера, и их размер равен размеру элемента, в фоне которого они расположены. Конечно, такое поведение не подходит для построения орнаментов.

К счастью, мы можем задать размер для градиента с помощью свойства background-size. Например, так:

background-size: 100px 200px;

Такая запись задаст всем фоновым изображениям ширину 100px и высоту 200px.

Задав размер для градиентов в фоне, мы получим фрагмент орнамента, повторением которого сможем управлять с помощью background-repeat.

На этом шаге мы увеличим контейнер, затем зададим размер для фоновых градиентов и отключим повторение, чтобы увидеть один кусочек орнамента.

.pattern {

width: 200px;

height: 200px;

margin: 100px auto;

background-image:

linear-gradient(135deg, white 25%, transparent 25%),

linear-gradient(225deg, white 25%, transparent 25%),

linear-gradient(45deg, white 25%, transparent 25%),

linear-gradient(315deg, white 25%, transparent 25%);

background-color: #333333;

box-shadow: 1px 1px 3px #333333;

background-size:100px 100px;

background-repeat:no-repeat;

}

## Строим сложный фон — финал [18/23]

На этом шаге мы завершим создание орнамента. Мы уже растянули контейнер на всё окно мини-браузера, чтобы орнамент было лучше видно. Осталось внести небольшие изменения в базовый фрагмент и размножить его.

Вернёмся к множественным фонам. Вы можете не только задавать несколько фоновых изображений, но и управлять размерами, расположением и повторением каждого из них по отдельности. Такая запись:

background-size: 100px 200px;

Задаст всем фоновым изображениям один и тот же размер. А такая запись:

background-size:

100px 200px,

200px 300px,

300px 400px;

Задаст всем фоновым изображениям разные размеры.

Чтобы задавать разные значения свойств для множественных фонов, вам нужно просто перечислять эти значения через запятую в том же порядке, в каком вы записали изображения.

1. **Цель 1**Уменьшите размер *всех* фоновых изображений до 50px по ширине и высоте.
2. **Цель 2**Затем для *первых двух* фонов задайте расположение 25px 0.
3. **Цель 3**Затем задайте *всем* фоновым изображениям повторение по горизонтали.
4. **Цель 4**И, наконец, включите им повторение во все стороны.

.pattern {

height: 100%;

background-image:

linear-gradient(135deg, white 25%, transparent 25%),

linear-gradient(225deg, white 25%, transparent 25%),

linear-gradient(45deg, white 25%, transparent 25%),

linear-gradient(315deg, white 25%, transparent 25%);

background-position:

25px 0,

25px 0,

0 0,

0 0;

background-size: 50px 50px,

50px 50px,

50px 50px,

50px 50px;

background-repeat: repeat-x;

background-color: #333333;

}

## Кнопки — часть 1 [19/23]

Градиенты могут применяться и в более прикладном плане. Например, для создания красивых кнопок.

В этой серии заданий мы с помощью градиентов создадим красивые фоны для нескольких кнопок.

А начнём с простейшей кнопки.

.simple-gray {

color: #333333;

background-image: linear-gradient(#ffffff, #dddddd);

}

## Кнопки — часть 2 [20/23]

Построим ещё одну кнопку с фоновым линейным градиентом.

Это будет стильная, почти плоская тёмно-красная кнопка. В этот раз будем использовать и колорстопы.

.strong-brown {

color: #ffffff;

background-image: linear-gradient(#cc0630 0%, #8f0222 75%, #6d0019 100%);

## Кнопки — часть 3 [21/23]

А сейчас создадим более сложную «блестящую и бликующую» псевдотрёхмерную кнопку.

Для этого нам понадобится чуть больше цветов, заданных вручную колорстопов и один резкий переход.

.glossy-blue {

color: #ffffff;

background-image:linear-gradient(#b8e1fc 0%, #90bcea 35%, #90bff0 50%, #6ba8e5 51%, #bdf3fd 100%);

}

## [Испытание: орнамент из сюрикенов [22/23]](http://forum.academy/t/ispytanie-ornament-iz-syurikenov-22-23/3740)

pattern {  
height: 100%;  
background-size: 115px 115px;  
background-image:  
linear-gradient(45deg, #555555 25px, transparent 25px),  
linear-gradient(315deg, #555555 25px, transparent 25px),  
linear-gradient(135deg, #555555 25px, transparent 25px),  
linear-gradient(225deg, #555555 25px, transparent 25px),  
linear-gradient(45deg, #000000 25px, transparent 25px),  
linear-gradient(225deg, #000000 25px, transparent 25px);  
background-repeat: repeat;  
background-position:  
70px -45px,  
-45px -80px,  
35px 70px,  
-80px 35px,  
35px -45px,  
-45px 35px;  
}

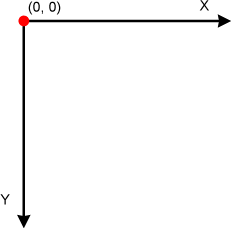
# Двумерные трансформации

## Перемещение по горизонтали [1/32]

В этом курсе мы рассмотрим возможности двухмерных трансформаций в CSS.

То, что раньше можно было сделать в окне браузера только с помощью JavaScript — плавное перемещение и масштабирование блоков, повороты и наклоны — теперь с лёгкостью реализуется на чистом CSS.

Для начала давайте разберёмся с системой координат, в которой перемещается объект:



её ключевой особенностью является то, что ось Y направлена вниз, а не вверх, так как веб-страница начинается с левого верхнего угла и идет вниз, а в CSS используется [обратная система координат](http://ru.wikipedia.org/wiki/%D0%9F%D1%80%D1%8F%D0%BC%D0%BE%D1%83%D0%B3%D0%BE%D0%BB%D1%8C%D0%BD%D0%B0%D1%8F_%D1%81%D0%B8%D1%81%D1%82%D0%B5%D0%BC%D0%B0_%D0%BA%D0%BE%D0%BE%D1%80%D0%B4%D0%B8%D0%BD%D0%B0%D1%82).

Первое перемещение, которое мы осуществим — горизонтальное. Мы будем двигать объекты влево и вправо.

Для этой и других трансформаций используется следующий синтаксис:

transform: [функция трансформации]([значение трансформации])

Горизонтальное перемещение осуществляется функцией translateX. Значение функции трансформации численное, а возможные единицы измерения — px, %, em или in.   
Например, такая функция переместит объект на 100 пикселей вправо по оси X:

transform: translateX(100px)

Кстати, знакомьтесь! Главного героя, которым вам предстоит управлять и трансформировать его заклинаниями окружающий мир, зовут Пендальф Синий. Вместе с ним вас ждёт увлекательное приключение…

.wizard {

top: 244px;

left: 30px;

transform:translateX(250px) translateX(-250px);

}

Курс «[Двумерные трансформации](https://htmlacademy.ru/courses/71)»

## Перемещение по вертикали [2/32]

Теперь давайте попробуем перемещать объекты по вертикали.

Для этого воспользуемся похожей на «горизонтальную» функцией translateY:

transform: translateY(-100px)

Такая функция переместит объект на 100 пикселей вертикально вверх. Заметьте, что для движения вверх используется отрицательное значение трансформации.

Как вы уже увидели в предыдущем задании, к объекту может быть одновременно применено несколько функций трансформации. При этом функции просто перечисляются через пробел после имени свойства transform, например:

transform: translateY(-100px) translateX(100px)

Такая трансформация переместит объект на 100 пикселей вправо и вверх по оси координат.

Давайте попробуем перенести мага по воздуху с помощью функций горизонтальных и вертикальных перемещений.

И конечно же надо не забыть прочитать заклинание левитации, чтобы волшебник мог лететь!

## Тренировка фаерболов [3/32]

В опасном мире фэнтези, где за каждым углом поджидает тролль или оборотень, Пендальф должен уметь постоять за себя. А лучшее средство атаки и защиты для любого уважающего себя боевого мага — это, как известно, фаерболы.

Давайте научимся бросать фаерболы по мишеням, чтобы в случае необходимости защитить Пендальфа от врагов.

Для этого применим изученные ранее заклинания перемещения: translateX, translateY, а также новое более лаконичное заклинание, объединяющее перемещение сразу по двум осям в одну функцию — translate:

transform: translate(перемещение по оси X [, перемещение по оси Y])

Как мы видим, два значения перемещения просто перечисляются через запятую, причём у них могут быть разные единицы измерения.

Значение перемещения по оси Y является необязательным аргументом, то есть если его не указать, а написать translate(перемещение по оси X), то значение перемещения по Y будет считаться равным 0 и функция будет работать аналогично translateX.

Приступим же наконец к огненным процедурам!

.fireball-1 {

transform:translate(310px);

}

.fireball-2 {

transform:translate(0,-190px);

}

.fireball-3 {

transform:translate(290px,-190px);

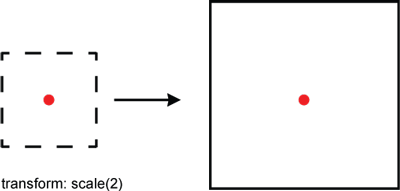
}

## Увеличение, уменьшение [4/32]

Ещё одной возможностью CSS-трансформаций является масштабирование блоков. С помощью функции scale можно увеличивать или уменьшать элементы.

Значением этой функции может быть любое число. При этом значение 1 считается точкой отсчёта, когда блок имеет оригинальные размеры. Рассмотрим примеры:

* scale(0.5) уменьшит объект в 2 раза;
* scale(2) увеличит объект в 2 раза;
* scale(0) полностью «схлопнет» объект, и его не будет видно;
* scale(1) оставит объект без изменений.



Вообще функция scale как и translate принимает 2 аргумента:

scale(масштаб-по-X [, масштаб-по-Y])

Если масштаб-по-Y не задан, то считается, что он такой же, как масштаб-по-X:

transform: scale(2) то же самое, что transform: scale(2, 2)

Также в случае, когда необходимо масштабировать объект только по одной оси независимо от другой, существуют функции scaleX(масштаб-по-Х) и scaleY(масштаб-по-Y).

.hippo {

transform:scale(0.25);

}

.mouse {

transform:scale(3);

}

## Защита города [5/32]

Вот и пришла пора впервые с пользой применить полученные знания: рано утром на город, где живёт Пендальф, напал волк!

В суматохе наш волшебник не успел наколдовать фаерболы, поэтому придётся в срочном порядке починить развалившуюся ограду.

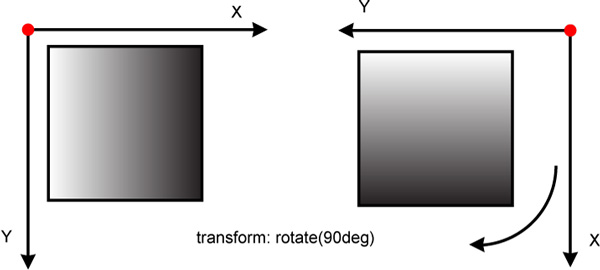
Для этого нужно собраться и быстро вспомнить всё, а также освоить пару новых приёмов.



В этом задании нам понадобится трансформация c функцией поворота transform: rotate. Её синтаксис: transform: rotate(угол наклона).

Положительное значение угла повернёт объект по часовой стрелке, а отрицательное — против. Единица измерения поворота — градусы deg, например: transform: rotate(180deg) повернёт объект на 180° по часовой стрелке, то есть перевернёт его.

При совместном применении трансформации поворота и перемещения есть небольшая тонкость. При повороте вместе с объектом на заданный угол поворачивается и вся его система координат:



То есть, чтобы повёрнутый на 90° блок двигать по горизонтали, нужно применять translateY, а по вертикали — translateX.

Перемещайте блок .fence-1:

1. **Цель 1**Поверните по часовой стрелке на 90°:   
   transform: rotate(90deg).
2. **Цель 2**Переместите вверх на 25px.
3. **Цель 3**И подвиньте вправо на 50px, совместив с первой частью ограды.

.fence-1 {

transform:rotate(90deg) translate(-25px, -50px);

## Начало путешествия [6/32]

Тревожные вести приходят с дальних рубежей королевства. Древнее зло проснулось в подземных пещерах. Выбора нет, кто-то должен остановить надвигающуюся опасность. И Пендальф отправляется в путь, вспоминая изученные заклинания, которые пригодятся в опасном путешествии.

Долгие дни дороги и безуспешные поиски приводят к большой горе, где он находит неприметный вход в подземные пещеры, где таится древнее зло. «Эх, была не была», — думает Пендальф и устремляется внутрь горы.

Впрочем, полученные знания пригождаются ему сразу же.

transform:translate(100px) scale(0.5) translate(540px);

## Лабиринт [7/32]

Из глубин пещеры доносятся страшные звуки, но пути к отступлению уже нет.

Сейчас нашему герою нужно бесстрашно пробраться через каменные проходы навстречу ужасным звукам.

Пендальф произносит заклинание левитации и начинает движение вперёд.

Подсказка: координаты ключевых точек задания кратны 10.

## Битва — часть 1 [8/32]

Пройдя лабиринт Пендальф выходит в пещеру с высоким сводом. Из темноты на него глядят две красные точки. «Кажется это не к добру», — думает Пендальф и зажигает фаербол…

В свете загоревшегося огня он видит страшное волосатое чудище, которое явно не обедало и заинтересованно смотрит на волшебника. Что делать — придётся защищаться!

.fireball {

transform:translate(250px, 90px);

}

.werewolf {

transform:rotate(90deg);

}

## Битва — часть 2 [9/32]

Во время ночлега в неуютной пещере Пендальф решает скоротать время и поучиться CSS-магии ещё немного.

Ещё одна возможность CSS-трансформаций — наклон объекта по осям X и Y под заданным углом. Наклоны создаются с помощью функций skewX и skewY.

Угол наклона задаётся в градусах — deg. Например:

transform: skewX(45deg)

transform: skewY(30deg)

Для оси X положительное значение угла наклоняет объект влево, а отрицательное — вправо. Для оси Y — вниз и вверх соответственно.

В примере ниже первый блок наклонён на 25° влево, а второй — на 25° вниз:

transform: skewX(25deg); transform: skewY(25deg);

Заметим, что наряду со skewX и skewY существует обобщающая функция skew, которая принимает два аргумента: skew(наклон-по-Х [, наклон-по-Y]). Значение наклона по оси Y является необязательным аргументом и по умолчанию равно 0. Но при этом поведение функции skew при трансформации отличается от одновременного применения skewX и skewY. Исторически так сложилось, что реализация skew в браузерах поддерживается сейчас только для совместимости прежнего контента, и даже исключалась из рабочего черновика спецификации CSS. В общем, вместо skew лучше пользоваться skewX и skewY.

И что бы вы думали? Как только Пендальф выучился новому заклинанию, к нему в гости пожаловал пещерный монстр, немного смахивающий на дракона. Что-то подсказало Пендальфу, что фаерболами убедить гостя уйти восвояси не выйдет…

.monster {

transform:skewX(30deg) skewY(30deg);

}

## Битва — часть 3 [11/32]

Итак, мы в штаб-квартире древнего зла. До победного конца остаётся самая малость — побороть главаря. Тут, к сожалению, фаерболами не обойдёшься и придётся призывать героя, который сможет навешать кренделей здоровяку…

Ну и конечно же не обойтись без изученных CSS-заклинаний!

.hero {

transform:scale(1.5) translate(40px);

}

.overlord {

transform:rotate(90deg);

}

## Финальная битва [13/32]

Перфекционизм никогда до добра не доводил. Не довёл и сейчас — собранный кристалл оказался ловушкой. Дух побеждённого злодея вселился в душу Пендальфа и телепортировал его в непонятную и неизвестную комнату.

А во время телепортации он успел изучить ещё одну интересную особенность transform: scale. А именно то, как ведёт себя объект при задании отрицательных числовых значений: при scaleX(-1), например, оригинальный объект зеркально отражается по горизонтали, а при scale(-1, -1) — отражается и по оси X, и Y. Большее, чем -1 значение ещё шире растянет отзеркалированный объект.

Ниже представлены примеры подобных трансформаций:

transform: scaleX(1);  
transform: scaleX(1); transform: scaleX(-1);  
transform: scaleX(-1); transform: scale(-1);  
transform: scale(-1); transform: scaleX(-1.5);  
transform: scaleX(-1.5);

Также в пути Пендальф успел прочитать брошюру «Битва с самим собой» и понял, что ему надо вызвать на дуэль своего двойника, но при этом никоим образом нельзя повреждать его, ведь тем самым он поразит себя. Как одолеть его — неизвестно, но пути назад уже не было, он решил искоренить древнее зло раз и навсегда…

.doppelganger {

visibility: visible;

transform: scaleX(-1);

}

.diamond {

transform: scale(0.6, -0.6) translate(-330px, 350px);

}

## Особенности transform-origin — часть 1 [14/32]

Теперь, когда древнее зло побеждено, а Пендальф добрался обратно домой, самое время разобраться с ещё одной тонкостью трансформаций, а именно со свойством transform-origin(а также повесить памятную фотокарточку на стену).

Это свойство используется совместно со свойством transform и задаёт точку отсчёта системы координат, в рамках которой будет работать трансформация.

Синтаксис transform-origin для двухмерных трансформаций следующий:

transform-origin: точка-отсчёта-по-X [, точка-отсчёта-по-Y]

Значения свойства задаётся в единицах измерения ширины в браузере (px, em…), в %, а также ключевыми словами left, right, top, bottom и center.

По умолчанию, значение transform-origin равно 50% 50%, то есть начало системы координат находится в центре объекта. Если не указывать значение точка-отсчёта-по-Y, то оно считается равным 50%.

Давайте разберёмся на примере функции scale, как ведёт себя трансформация при различных значениях transform-origin: в данном случае блок будет менять размер относительно заданной точки.

## Особенности transform-origin — часть 2 [15/32]

А теперь попробуем сделать трансформацию плавной. Для этого в CSS предусмотрено свойство transition. Не будем пока вдаваться в подробности, как именно работает и применяется transition — это будет подробно рассмотрено в ближайшем курсе.

Если в двух словах, transition позволяет изменить значение какого-либо свойства плавно. В нашем случае плавно будет меняться свойство transform с функцией scale, а переход между двумя значениями будет длиться полсекунды.

Всё это наглядно покажет, как при разных значениях transform-origin меняется точка начала системы координат. Для демонстрации этого давайте задавать значения transform-origin и нажимать кнопку «Запустить», по которой для блока .picture переключается класс active

## Особенности transform-origin — часть 3 [16/32]

А теперь давайте посмотрим, как меняется поведение трансформации поворота transform: rotate в зависимости от разных значений transform-origin.

В данном случае будет изменяться расположение оси вращения элемента.

## Особенности transform-origin — часть 4 [17/32]

На статичном примере не так хорошо видно, как изменяется ось вращения элемента, когда мы изменяем transform-origin.

Поэтому мы сделали ещё одно задание, где элемент будет прокручиваться на 360° при нажатии кнопки. И здесь будет отлично видно, как изменяется ось вращения.

.picture {

transition: transform 1s ease;

transform: rotate(0deg);

transform-origin:50% 100%;

}

.picture.active {

transform: rotate(360deg);

}

## Центровка с помощью transform: translate [18/32]

Для центровки одного блока внутри другого блока обычно используют [классический трюк](http://css-tricks.com/quick-css-trick-how-to-center-an-object-exactly-in-the-center/) с позиционированием, относительными координатами и отрицательными маргинами.

Этот приём хорошо работает, когда центруемый блок имеет фиксированные размеры, но если его размеры могут изменяться, то возникают проблемы.

С помощью трансформаций можно решить эту проблему и центровать блоки с переменными размерами. Делается это с помощью translate.

.picture {

position: relative;

width: 50%;

height: 50%;

left:50%;

top:50%;

transform:translateX(-50%) translateY(-50%);

}

## Поворот текста в блоках [19/32]

На этом приключения волшебника Пендальфа заканчиваются, поэтому давайте вернёмся в реальный мир и рассмотрим техники создания элементов интерфейсов и декоративных эффектов при помощи CSS-трансформаций.

В этом задании мы перевернём заголовок статьи и поместим его сбоку. Для этого воспользуемся уже знакомыми свойствами transform-origin и transform: rotate.

1. **Цель 1**Для h1 задайте абсолютное позиционирование, координаты top — 10px, left — 0.
2. **Цель 2**Затем article задайте сплошную рамку слева border-leftтолщиной 40px цвета #7fdbff.
3. **Цель 3**Для h1 задайте transform-origin со значением 0 0 и поверните его на 90° по часовой стрелке.

## Поворот текста в фоне [20/32]

Вот ещё один интересный эффект, создаваемый с помощью трансформации поворота.

На этот раз мы будем поворачивать «фоновый» текст, а также немного наклоним блок с заголовком функцией skew. А чтобы наклонным был только блок, но не текст внутри, — наклоним текст в обратную сторону на тот же угол.

1. Заголовку h1 задайте наклон на 15° вправо.
2. **Цель 2**А для span внутри заголовка — наклон влево на 15°.
3. **Цель 3**Поверните p внутри article на 15° против часовой стрелки.

## Нестандартные тени [21/32]

Иногда бывает нужно реализовать тени, отличающиеся от стандартных box-shadow, без применения картинок.

Например, тени, наклонённые в разные стороны.

Сделать это с помощью box-shadow невозможно. Поэтому нужно использовать более сложный приём:

* c помощью псевдоэлементов before и after создаём два блока с обычными тенями;
* наклоняем эти блоки с помощью rotate;
* задаём им отрицательный z-index.

Это переместит блоки с тенями под родительский контейнер так, что наружу будут выглядывать только кусочки теней.

1. Псевдоэлемент before блока shadow-box поверните на 3°против часовой стрелки.
2. **Цель 2**Затем псевдоэлемент after — по часовой на 3°.
3. **Цель 3**Задайте обоим пседвоэлементам z-index равный -1.

.shadow-box::before,

.shadow-box::after {

content: "";

position: absolute;

top: 80%;

bottom: 15px;

width: 45%;

max-width: 300px;

background: #dddddd;

box-shadow: 0 15px 10px #555555;

}

.shadow-box::before {

z-index:-1;

left: 10px;

transform:rotate(-3deg);

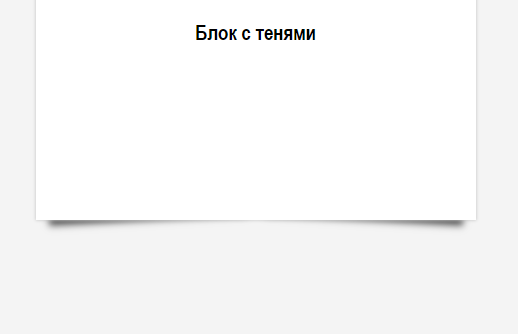
}

.shadow-box::after {

z-index:-1;

right: 10px;

transform:rotate(3deg);

}

## Эффекты при наведении: кнопки — часть 1 [22/32]

Довольно много интересных эффектов можно добиться с помощью трансформаций при реализации разных элементов интерфейса, например, кнопок. Им можно добавить немного динамики, используя transform совместно с transition для создания простейшей плавной анимации.

В следующих нескольких примерах мы создадим эффекты при наведении на кнопку. Во всех случаях будет применяться свойство transition для плавной смены значений трансформации или прозрачности. Детально тема плавных переходов будет разбираться в ближайших курсах.

В этом задании при наведении на кнопку давайте будем динамически оборачивать вокруг своей оси иконку и одновременно немного её увеличивать.

Добавьте для правила .btn-green:hover .icon:

1. **Цель 1**Трансформацию поворота на 360° по часовой стрелке.
2. **Цель 2**И трансформацию масштабирования со значением 1.2.

После каждого изменения наводите курсор на зелёную кнопку для проверки.

.btn-green .icon {

width: 24px;

height: 24px;

transition: all 0.4s ease-in-out;

}

.btn-green:hover .icon {

transform: rotate(360deg) scale(1.2);

}

## Эффекты при наведении: кнопки — часть 2 [23/32]

В этом задании создадим вторую кнопку с немного другим эффектом у иконки: при наведении иконка должна увеличиваться и одновременно становиться полностью прозрачной, а на её месте синхронно будет появляться дополнительный скрытый блок с ценой .hidden.

Здесь будет использован плавный переход между двумя значениями прозрачности opacity и transform: scale.

1. **Цель 1**Для .btn-yellow:hover .hidden задайте opacity равную 1.
2. **Цель 2**Для .btn-yellow:hover .icon задайте трансформацию scaleсо значением 10.
3. **Цель 3**А также opacity равную 0.

После каждого изменения наводите курсор на жёлтую кнопку для проверки.

.btn-yellow .icon {

width: 32px;

height: 32px;

transition: all 0.4s ease-in-out;

}

.btn-yellow:hover .hidden {

left: 20px;

transition: opacity 0.2s ease-in-out;

opacity:1;

}

.btn-yellow:hover .icon {

transform:scale(10);

opacity:0;

}

## Эффекты при наведении: кнопки — часть 3 [24/32]

Ещё одна кнопка и ещё один эффект, похожий на предыдущие: при наведении давайте будем плавно скрывать иконку свойством opacity и одновременно показывать скрытый блок .hidden. При этом можно ещё дополнить эффект тем, что скрытый блок будет изначально «схлопнут» свойством scale до нуля, и при появлении он будет увеличиваться вместе с увеличением непрозрачности.

1. **Цель 1**Для .btn-blue:hover .icon задайте opacity равную 0.
2. **Цель 2**Для .btn-blue:hover .hidden задайте трансформацию scaleсо значением 1 и opacity равную 1.
3. **Цель 3**Для .btn-blue .hidden задайте трансформацию scale со значением 0.

После каждого изменения наводите курсор на синюю кнопку для проверки.

.btn-blue .icon {

width: 28px;

height: 28px;

transition: all 0.2s ease-in-out;

}

.btn-blue .hidden {

left: 15px;

top: 12px;

opacity: 0;

transform:scale(0);

}

.btn-blue .hidden img {

width: 28px;

height: 28px;

}

.btn-blue:hover .hidden {

transition:

opacity 0.2s ease-in-out,

transform 0.2s ease-in-out 0.1s;

transform:scale(1);

opacity:1;

}

.btn-blue:hover .icon {

opacity:0;

}

## Эффекты при наведении: галерея [25/32]

В любой ситуации, когда стоит задача плавно менять позицию или размеры блока, а также каким-либо способом менять его внешний вид, встаёт выбор: пользоваться связкой transition + transform или же задействовать javascript и динамически менять ширину/высоту, координаты объектов в скрипте.

Зачастую выбор падает на js-реализацию с помощью библиотеки jQuery, но в большинстве случаев «чистая» CSS-реализация простых визуальных эффектов намного производительнее аналогичных jQuery-функций. Кроме того, в ряде случаев определённые трансформации в CSS могут для повышения производительности задействовать не только центральный процессор компьютера или мобильного девайса, но и ресурсы графического адаптера, что позволяет разгрузить процессор и избавиться от «тормознутости» эффектов.

В этом задании с помощью трансформации и плавных переходов давайте реализуем простую мини-галерею картинок, которые увеличиваются и становятся полупрозрачными при наведении.

.deck img {

width: 100px;

transition: all 0.1s linear;

}

.deck img:hover {

opacity:0.6;

transform:scale(1.1);

}

## «Стопка» карт [26/32]

В этом задании давайте создадим «стопку» карт. Без эффектов эти картинки абсолютно спозиционированы и располагаются слоями друг под другом. Наша задача — немного повернуть нижележащие карты, чтобы они выглядывали из-под верхних. Как обычно воспользуемся transform: rotate.

.deck img {

position: absolute;

transition: all 0.1s linear;

transform-origin: 0 0;

}

.queen {

transform:rotate(-10deg);

}

.jack {

transform:rotate(-20deg);

}

.ten {

transform:rotate(-30deg);

}

## Круговое меню, подготовка [27/32]

В этой серии заданий мы построим сложный элемент интерфейса, в котором ключевую роль играют трансформации. Это круговое меню.

Разметка меню не отличается от обычной. Внутри тэга <nav> расположен обычный список <ul>с несколькими элементами списка со ссылками.

Мы разберём подход на двух пунктах меню, а потом добавим остальные по аналогии.

Исходные стили для списка и его пунктов уже подготовлены, а также создан элемент, который обозначает центр меню.

Для того, чтобы вращать элементы списка в меню, нужно правильно расположить их и задать им ось вращения:

* ось должна находиться в правом нижнем углу элемента списка;
* этот угол должен совпадать с центром меню.

Для решения этой задачи используем свойство transform-origin, а также свойства позиционирования.

Кстати, чтобы меню получилось ровным, его пункты должны немного выходить за границы контейнера.

1. **Цель 1**Задайте элементам списка абсолютное позиционирование.
2. **Цель 2**Свойства top и left со значением -10px.
3. **Цель 3**А также transform-origin со значением 100% 100%.

<!DOCTYPE html>

<html lang="ru">

<head>

<title>Круговое меню, подготовка</title>

<meta charset="utf-8">

<base href="/assets/icons/">

</head>

<body>

<nav class="circle-menu">

<div class="center"></div>

<ul>

<li><a href="#"></a></li>

<li><a href="#"></a></li>

</ul>

</nav>

</body>

</html>

html,

body {

margin: 0;

padding: 0;

background: #f5f5f5;

}

.circle-menu {

position: relative;

width: 400px;

height: 400px;

margin: 50px auto;

background: white;

box-shadow: 0 0 3px #cccccc;

}

.circle-menu ul {

position: absolute;

width: 300px;

height: 300px;

margin: 50px;

padding: 0;

list-style: none;

box-shadow: 0 0 0 1px #3498db;

}

.circle-menu .center{

position: absolute;

top: 195px;

left: 195px;

z-index: 1000;

border: 5px solid #34495e;

border-radius: 50%;

box-shadow: 0 0 3px #cccccc;

}

.circle-menu li {

width: 160px;

height: 160px;

background: rgba(145, 145, 145, 0.1);

box-shadow: inset 0 0 1px #cccccc;

position:absolute;

top:-10px;

left:-10px;

transform-origin:100% 100%;

}

## Круговое меню, шаг 1 [28/32]

Теперь можно раcсчитывать углы секторов меню и углы поворота пунктов. Используем такие формулы:

[угол сектора] = 360° / [количество пунктов меню]

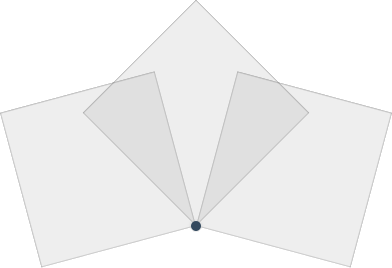
[поворот пункта n] = (n - 1) \* [угол сектора]

В нашем меню может быть максимум 6 пунктов, поэтому для него получатся такие значения:

[угол сектора] = 360° / 6 = 60°

[поворот 1 пункта] = (1 - 1) \* 60° = 0°

[поворот 2 пункта] = (2 - 1) \* 60° = 60°



Для вращения пунктов, мы используем rotate. Но одного вращения недостаточно, так как элементы списка будут пересекаться, ведь их угол больше 60°.

Чтобы исправить это, мы искривим пункты с помощью skew. А угол наклона посчитаем так:

[угол наклона] = 90° - [угол сектора]

В нашем случае получится 30°.

Итого: первый элемент списка нужно наклонить на 30°, а второй — повернуть на 60° и наклонить на 30°.

Кстати, возможны две записи:

transform: rotate(60deg) skew(30deg); // 1 вариант

transform: skew(30deg) rotate(60deg); // 2 вариант

Их эффект будет отличаться, так как порядок трансформаций важен. И нам нужен первый вариант.

1. **Цель 1**Наклоните первый элемент списка на 30°.
2. **Цель 2**Поверните второй элемент списка на 60° и наклоните на 30°.

.circle-menu li:nth-child(1) {

transform: skew(30deg);

}

.circle-menu li:nth-child(2) {

transform: rotate(60deg) skew(30deg);

}

## Круговое меню, шаг 2 [29/32]

Внутри пунктов меню расположены ссылки. Для них задан блочный тип, размеры и фон — эти исходные стили нужно просто раскомментировать.

На предыдушем шаге мы повернули и скосили пункты меню, но при этом скосилось и их содержимое, то есть ссылки. Чтобы вернуть им нормальный вид, нужно их «разнаклонить» и развернуть обратно.

«Разнаклонять» будем на противоположный угол, то есть на -30°.

Расчёт угла разворота сложнее, вот формула:

[угол разворота] = -1 \* (90° – ([угол сектора] / 2))

В нашем случае получится:

[угол разворота] = -1 \* (90° - (60° / 2)) = -60°

Обратите внимание на порядок трансформаций для ссылок: сначала skew, потом rotate.

И ещё одна важная деталь. Нам нужно сместить ссылки так, чтобы они располагались во внутренней половине пунктов меню. Для этого используем маргины.

1. **Цель 1**Раскомментируйте CSS-правило для ссылок в меню.
2. **Цель 2**Наклоните ссылки на -30°.
3. **Цель 3**А затем дополнительно поверните их на -60°.
4. **Цель 4**Добавьте ссылкам маргины сверху и слева 40px.

.circle-menu li a {

display: block;

width: 160px;

height: 160px;

font-size: 0;

background: rgba(241, 196, 15, 0.5) url("monitor-4x.png") no-repeat 50% 40%;

transform:skew(-30deg) rotate(-60deg);

margin-top:40px;

margin-left:40px;

}

## Круговое меню, шаг 3 [30/32]

Промежуточный результат не очень похож на круглое меню, но ещё несколько строчек CSS-кода и всё изменится.

Первым делом мы обрежем части ссылок, которые выходят за пределы пунктов меню, и избавимся от избыточных пересечений. Для этого зададим свойство overflow со значением hidden для элементов списка.

Затем нужно будет придать меню круглую форму.

Для этого будем работать с самим списком ul, который сейчас выглядит как квадрат с синей рамкой. Добавим ему свойство border-radius со значением 50%, и квадрат превратится в круг.

А дальше останется только обрезать всё, выходящее за пределы круга. И снова используем overflow.

Круглое меню из двух пунктов готово, можно убрать вспомогательные фоны и рамки.

1. **Цель 1**Задайте для элементов списка overflow: hidden.
2. **Цель 2**Задайте списку border-radius: 50%.
3. **Цель 3**А затем тоже overflow: hidden.
4. **Цель 4**Удалите тени и фоны у списка и элементов списка.

html,

body {

margin: 0;

padding: 0;

background: #f5f5f5;

}

.circle-menu {

position: relative;

width: 400px;

height: 400px;

margin: 50px auto;

background: white;

box-shadow: 0 0 3px #cccccc;

}

.circle-menu ul {

position: absolute;

width: 300px;

height: 300px;

margin: 50px;

padding: 0;

list-style: none;

border-radius:50%;

overflow:hidden;

}

.circle-menu .center {

position: absolute;

top: 195px;

left: 195px;

z-index: 1000;

border: 5px solid #34495e;

border-radius: 50%;

box-shadow: 0 0 3px #cccccc;

}

.circle-menu li {

position: absolute;

top: -10px;

left: -10px;

width: 160px;

height: 160px;

transform-origin: 100% 100%;

overflow:hidden;

}

.circle-menu li:nth-child(1) {

transform: rotate(0deg) skew(30deg);

}

.circle-menu li:nth-child(2) {

transform: rotate(60deg) skew(30deg);

}

.circle-menu li a {

display: block;

width: 160px;

height: 160px;

margin-top: 40px;

margin-left: 40px;

font-size: 0;

background: rgba(241, 196, 15, 0.5) url("monitor-4x.png") no-repeat 50% 40%;

transform: skew(-30deg) rotate(-60deg);

}

## Круговое меню, финал [31/32]

Вот и вся техника для создания круглого меню:

1. Переносим ось вращения пунктов в угол с помощью transform-origin и совмещаем этот угол с центром меню.
2. Наклоняем и поворачиваем пункты меню, используя формулы:
3. [угол сектора] = 360° / [количество пунктов меню]
4. [поворот пункта n] = (n - 1) \* [угол сектора]

[угол наклона] = 90° - [угол сектора]

1. «Разнаклоняем» и разворачиваем содержимое пунктов меню:
2. [угол наклона содержимого] = -1 \* [угол наклона пункта]

[угол разворота] = -1 \* (90° – ([угол сектора] / 2))

1. Смещаем содержимое пунктов к центру круга.
2. Закругляем контейнер меню с помощью border-radius и обрезаем всё лишнее с помощью overflow: hidden.

А после этих шагов можно применять дополнительное оформление. Например, задавать разные фоны пунктов, рамку для контейнера меню и так далее.

Давайте напоследок добавим третий пункт меню, чтобы у нас получился полукруг. Угол поворота третьего пункта будет равен 120°:

Серия заданий сделана по мотивам этой статьи [Building a Circular Navigation with CSS Transforms](http://tympanus.net/codrops/2013/08/09/building-a-circular-navigation-with-css-transforms/). Примеры были адаптированы и упрощены для курса.

1. **Цель 1**Добавьте в меню третий элемент списка со ссылкой внутри.
2. **Цель 2**Поверните третий элемент списка на 120°.
3. **Цель 3**А затем наклоните его на 30°.

<!DOCTYPE html>

<html lang="ru">

<head>

<title>Круговое меню, финал</title>

<meta charset="utf-8">

<base href="/assets/icons/">

</head>

<body>

<nav class="circle-menu">

<div class="center"></div>

<ul>

<li><a href="#">monitor</a></li>

<li><a href="#">aperture</a></li>

<li><a gref="#">audio</a></li>

</ul>

</nav>

</body>

</html>

html,

body {

margin: 0;

padding: 0;

background: #f5f5f5;

}

.circle-menu {

position: relative;

width: 400px;

height: 400px;

margin: 50px auto;

background: white;

box-shadow: 0 0 3px #cccccc;

}

.circle-menu ul {

position: absolute;

width: 300px;

height: 300px;

margin: 50px;

padding: 0;

list-style: none;

border-radius: 50%;

overflow: hidden;

}

.circle-menu .center{

position: absolute;

top: 195px;

left: 195px;

z-index: 1000;

border: 5px solid #34495e;

border-radius: 50%;

box-shadow: 0 0 3px #cccccc;

}

.circle-menu li {

position: absolute;

top: -10px;

left: -10px;

width: 160px;

height: 160px;

transform-origin: 100% 100%;

overflow: hidden;

}

.circle-menu li:nth-child(1) {

transform: rotate(0deg) skew(30deg);

}

.circle-menu li:nth-child(2) {

transform: rotate(60deg) skew(30deg);

}

.circle-menu li:nth-child(3) {

transform: rotate(120deg) skew(30deg);

}

.circle-menu li a {

display: block;

width: 160px;

height: 160px;

margin-top: 40px;

margin-left: 40px;

font-size: 0;

background: rgba(241, 196, 15, 0.5) url("monitor-4x.png") no-repeat 50% 40%;

transition: background-color 0.5s;

transform: skew(-30deg) rotate(-60deg);

}

.circle-menu li:nth-child(even) a {

background-color: rgba(241, 196, 15, 0.75);

}

.circle-menu li a:hover {

background-color: rgba(241, 196, 15, 1);

}

/\* Иконки \*/

.circle-menu li:nth-child(2) a {

background-image: url("aperture-4x.png");

}

.circle-menu li:nth-child(3) a {

background-image: url("audio-4x.png");

}

.circle-menu li:nth-child(4) a {

background-image: url("battery-empty-4x.png");

}

.circle-menu li:nth-child(5) a {

background-image: url("bluetooth-4x.png");

}

.circle-menu li:nth-child(6) a {

background-image: url("browser-4x.png");

}

# Оформление текста, часть 2

## Unicode-символы [1/26]

В этом курсе мы продолжим изучать возможности по оформлению текста с помощью CSS, которые не были рассмотрены в [первом курсе](https://htmlacademy.ru/courses/43).

Мы будем создавать подписи к фотографиям и красивые текстовые элементы дизайна, а также рассмотрим приёмы оформления длинных текстов.

А для начала давайте создадим небольшой комикс. Он будет состоять всего из четырёх графических символов, которые познакомят нас с историей главного героя комикса — смайлика.

Но где нам взять графические символы? Тут на помощь придёт кодировка Unicode. Помимо тех символов, которые помещаются у нас на клавиатуре, в стандарте кодирования Unicode представлены знаки почти всех письменных языков, а также разные специальные символы. В разметке HTML возможно написать любой символ Unicode, используя специальные ссылки-мнемоники: в виде слова, например, &название; или в числовом обозначении с использованием десятичного или шестнадцатиричного кода, например, &#224;.

Примеры Unicode-символов:

|  |  |  |
| --- | --- | --- |
| &copy; | &#169; | © |
| &alpha; | &#945; | α |
| &para; | &#182; | ¶ |

Есть [специальный сервис](http://copypastecharacter.com/), где можно найти и скопировать нужный символ.

С теорией разобрались, давайте же создадим комикс про приключения смайлика.

## Свойство text-shadow [2/26]

Сначала познакомимся с текстовыми тенями.

Тени для текста создаются с помощью CSS-свойства text-shadow. Оно очень похоже на box-shadow, рассмотренное в [одном из предыдущих курсов](https://htmlacademy.ru/courses/51), но имеет свои особенности:

* свойство text-shadow применяется к тексту;
* форма тени повторяет форму текстовых символов;
* можно управлять смещением тени, её цветом, а также размытием;
* нельзя управлять растяжением текстовой тени;
* можно создавать множественные тени.

Для начала давайте создадим тень у заголовка и смайлика, а потом рассмотрим все параметры свойства подробнее.

## Смещение текстовой тени по горизонтали [3/26]

Рассмотрим подробнее параметры текстовой тени:

text-shadow: 0px 0px 0px #333333;

Первый параметр свойства text-shadow обязателен и задаёт смещение тени по горизонтали относительно текста. Положительное значение этого параметра сдвигает тень вправо, отрицательное — влево.

Смещение задаётся в абсолютных и относительных единицах измерения: px, em, pt и других.

Попробуем поменять значение первого параметра.

Курс «[Оформление текста, часть 2](https://htmlacademy.ru/courses/79)»

## Смещение текстовой тени по вертикали [4/26]

text-shadow: 0px 0px 0px #333333;

Второй параметр свойства text-shadow тоже обязателен и задаёт смещение тени по вертикали. Положительное значение сдвигает тень вниз, отрицательное — вверх.

Изменим значение второго параметра.

## Размытие текстовой тени [5/26]

text-shadow: 0px 0px 0px #333333;

Третий параметр свойства text-shadow необязателен и задаёт радиус размытия тени. Если этот параметр не задан, то по умолчанию устанавливается равным 0. Чем больше значение параметра, тем сильнее тень размывается и становится светлее.

Изменим радиус размытия тени.

## Цвет текстовой тени [6/26]

text-shadow: 0px 0px 0px #333333;

Последний параметр свойства text-shadow необязателен и задаёт цвет тени.

Если цвет тени не указан, то он совпадает с цветом текста. Цвет задаётся в любом цветовом формате, может быть полупрозрачным.

Попробуем задавать разные цвета теням.

## Эффект вдавленного текста [7/26]

У текста может быть задано сразу несколько теней. Для этого определения теней нужно перечислить через запятую. При этом тени распределяются по правилу: первая тень в списке — самая верхняя, последняя в списке — самая нижняя.

text-shadow:

1px 1px 1px #111111,

2px 2px 2px #222222;

С помощью множественных теней можно добиваться необычных результатов. Например, можно добавить тексту эффект «вдавленности» с помощью двух однотонных теней: более тёмная смещается немного вверх и влево, а более светлая — вниз и вправо.

Давайте реализуем это!

Курс «[Оформление текста, часть 2](https://htmlacademy.ru/courses/79)»

## Декоративная ретро-тень [8/26]

Ещё один текстовый эффект создаётся из двух резких теней.

Тени смещены в одну сторону. Нижняя тень смещена чуть сильнее и её цвет отличается от цвета фона, а верхняя тень смещена слабее и цвет её совпадает с цветом фона. Получается интересный эффект обводки.

Такой эффект хорошо подходит для заголовков с винтажным стилем.

1. **Цель 1**

Заголовку h1 задайте тень с горизонтальным и вертикальным смещением 4px без размытия с цветом rgba(0, 0, 0, 0.2),

1. **Цель 2**а затем добавьте перед ней ещё одну тень с горизонтальным и вертикальным смещением 2px без размытия с цветом #e5d4c0.
2. Курс «[Оформление текста, часть 2](https://htmlacademy.ru/courses/79)»

## Логотип: нестандартный шрифт [10/26]

1. Помимо теней, необычных текстовых эффектов можно добиться с помощью подключения нестандартных шрифтов. В [курсе про HTML5](https://htmlacademy.ru/courses/73/run/8%22) уже рассматривался пример подключения шрифтов с помощью конструкции @font-face.
2. В этом и следующих двух заданиях мы создадим логотип вымышленной компании «Warm photo» с использованием веб-шрифтов.
3. Для начала подключим шрифт для надписи логотипа и используем его в CSS.
4. **Цель 1**Раскомментируйте подключение веб-шрифта @font-face,
5. **Цель 2**а затем задайте ссылке шрифт "Montserrat Alternates", sans-serif размером 55px.

@font-face {

font-family: "Montserrat Alternates";

src:

local("MontserratAlternates-Regular"),

url("fonts/MontserratAlternates.woff") format("woff");

}

## Логотип: иконочный шрифт [11/26]

Помимо букв шрифт может содержать и другие графические символы. Существуют инструменты, позволяющие собрать свой собственный шрифт из SVG-объектов. Их довольно много: [icomoon.io/app](https://icomoon.io/app/), [fontastic.me](http://fontastic.me/), [glyphter.com](http://glyphter.com/) и другие.

Для простоты и наглядности в этом задании мы используем сервис Glyphter, чтобы создать шрифт с единственным символом, который будет храниться по адресу латинской буквы A.

Затем мы загрузим и подключим наш шрифт в CSS, напишем в коде букву A, но вместо неё увидим в браузере иконку.

Полученная иконка и будет нашим графическим символом в логотипе.

Преимуществом шрифтовых иконок перед растровыми картинками является то, что их легко смасштабировать или перекрасить в другой цвет с помощью CSS-свойств, применимых к тексту. Также на экранах с большой плотностью пикселей шрифт не «замыливается», как это происходит с обычными картинками.

1. **Цель 1**Раскомментируйте подключение иконочного шрифта Glyphter,
2. **Цель 2**затем в ссылке перед надписью Warmphoto вставьте span с символом A
3. **Цель 3**и задайте спану шрифт Glyphter.

@font-face {

font-family: "Glyphter";

src: url("fonts/Glyphter.woff") format("woff");

font-weight: normal;

font-style: normal;

}

## Логотип: выравнивание размеров letter-spacing [12/26]

Для завершения оформления логотипа нам остаётся задать символам нужные размеры и отступы.

Давайте сделаем надпись «WARMPHOTO» плотнее, уменьшив межсимвольное расстояние.

Это делается c помощью свойства letter-spacing. Браузер по умолчанию устанавливает межсимвольное расстояние автоматически согласно определенному значению у каждого шрифта. Значение по умолчанию соответствует letter-spacing: 0.

Увеличение или уменьшение значения letter-spacing изменит расстояние между символами на заданную величину. Значение задаётся в px, em, pt или других единицах длины CSS.

1. **Цель 1**Задайте ссылке межсимвольное расстояние -3px.
2. **Цель 2**Спану внутри ссылки задайте вертикальное выравнивание middle,
3. **Цель 3**а затем margin справа 18px и размер шрифта 78px.



## Декоративная стилизация строки, шаг 1 [13/26]

Давайте создадим ещё один интересный элемент дизайна. Стилизуем надпись «WARMPHOTO» с помощью подключаемого шрифта, размеров и отступов так, чтобы буквы сместились на новую строку и образовали квадрат.

Сначала создадим рамку и зададим тексту нужный шрифт и размер.

Блоку .square задайте:

1. **Цель 1**сплошную рамку толщиной 5px и цветом #333333;
2. **Цель 2**шрифт "Montserrat Alternates", sans-serif размером 72px;
3. **Цель 3**ширину 260px.

## Декоративная стилизация строки, шаг 2 [14/26]

Теперь нам нужно изменить правила переноса, чтобы символы, не поместившиеся по ширине в контейнер, переносились на новую строку.

Для этого в CSS существует свойство word-wrap. Оно принимает значения: normal и break-word. В случае normal слова переносятся на новую строку по обычным правилам (то есть по пробелам). А при значении break-word перенос производится браузером в тех местах, где слова перестают помещаться в контейнер.

Это как раз то, что нам нужно: зададим нужный перенос слова, увеличим межсимвольное расстояние и поправим отступ.

Также нужно заметить, что свойство word-wrap — изначально проприетарное расширение Microsoft, а в текущем черновике спецификации CSS3 переименовано в overflow-wrap. Название word-wrap сейчас обозначено как «альтернативное» именование.

Блоку .square задайте:

1. **Цель 1**правило переноса слова word-wrap со значением break-word;
2. **Цель 2**расстояние между символами равное 24px;
3. **Цель 3**padding слева равный 24px.



## Свойство text-indent [15/26]

Рассмотрим свойства, с помощью которых можно улучшить читабельность или украсить большие тексты.

И первым свойством будет text-indent. Оно устанавливает отступ для первой строки блока текста. С помощью этого свойства удобно делать красную строку в абзацах.

Значение свойства может принимать положительное или отрицательное значение, задаётся в px, em, pt или других единицах длины CSS. При задании значения в процентах, отступ первой строки вычисляется в зависимости от ширины блока.

## Псевдоэлемент ::first-letter [16/26]

В CSS существуют особые псевдоэлементы, позволяющие стилизовать первую букву в слове или первую строку в блоке текста.

Крупная, отличная от прочих, первая буква блока текста называется буквицей. Исторически буквицей украшали главы и разделы печатных книг и рукописей. Для создания буквицы воспользуемся псевдоэлементом ::first-letter:

p::first-letter { color: red; }

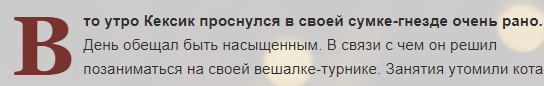
Давайте создадим и оформим буквицу.

В браузере Mozilla Firefox существует проблема с отрисовкой применённых стилей для псевдоэлемента ::first-letter. Поэтому, чтобы увидеть результат, вы можете делать задание в Chrome. Проверка сработает и в Mozilla.

1. **Цель 1**Псевдоэлементу ::first-letter первого параграфа задайте float: left,
2. **Цель 2**а затем шрифт "Georgia", serif размером 76px и межстрочный интервал 60px.

## Псевдоэлемент ::first-line [17/26]

1. Первую строку блока текста можно стилизовать при помощи псевдоэлемента ::first-line.
2. p::first-line { color: green; }
3. Мы добавили буквице небольшой padding для выравнивания её относительно текста. Давайте завершим стилизацию буквицы и оформим первую строку первого параграфа.
4. В браузере Mozilla Firefox существует проблема с отрисовкой применённых стилей для псевдоэлементов ::first-letter и ::first-line. Поэтому, чтобы увидеть результат, вы можете делать задание в Chrome. Проверка сработает и в Mozilla.
5. **Цель 1**Псевдоэлементу ::first-letter первого параграфа задайте цвет текста #8b3333,
6. **Цель 2**затем обнулите text-indent первого параграфа,
7. **Цель 3**а потом для псевдоэлемента ::first-line первого параграфа задайте полужирный шрифт.



## Колоночная разметка: свойство column-count [18/26]

Многоколоночная разметка CSS — замечательная возможность, позволяющая разбить блок с текстом на несколько колонок. Обычно очень длинные строки сложно читать: если приходится слишком долго перемещать взгляд с конца одной строки на начало другой, можно легко потерять нужную строку. Чтобы сэкономить место на экране, но при этом сделать текст читабельным, можно разбить один сплошной блок текста на несколько колонок, как это делается в газетах.

Уже сейчас во всех популярных современных браузерах можно использовать многоколоночную разметку CSS. Давайте рассмотрим основные свойства, позволяющие создавать колонки.

Первое из них — column-count. Это свойство принимает в качестве параметра целое число и делит блок текста на заданное число колонок, равных друг другу по ширине.

Многоколоночная разметка CSS уже поддерживается в [этих браузерах](http://caniuse.com/#feat=multicolumn).

## Колоночная разметка: свойство column-width [19/26]

Свойство column-width задаёт минимальную желаемую ширину колонки. Если свойство column-count ещё не было задано, браузер автоматически поделит текст на такое количество колонок, чтобы они уместились во всю доступную ширину.

Значение свойства положительное и задаётся в px, em, pt или других единицах длины CSS.

Следует отметить, что если одиночная строка может включать от 45 до 75 символов, чтобы быть читабельной, то для колонок текста рекомендуется придерживаться ширины, включающей 40-50 символов.

## Колоночная разметка: свойство column-gap [20/26]

Между колонками есть промежуток. Рекомендованная ширина промежутка по умолчанию равна 1em.

Но она может быть изменена при помощи свойства column-gap. Единицы измерения тоже px, em, pt и другие.

## Направление текста [21/26]

Ещё одной интересной, но малоиспользуемой возможностью работы с текстом в CSS является управление направлением текста. Обычно эта возможность применяется для корректного отображения арабского языка и иврита, в которых чтение происходит справа налево. В CSS за направление текста отвечает свойство direction, принимающее значения ltr (направление слева направо) и rtl (направление справа налево).

Однако помимо направления текста свойство direction также влияет на позицию полосы прокрутки в блоке.

Если попробовать задать для кириллицы или латиницы обратное направление текста (справа налево) direction: rtl;, мы увидим, что текст в блоке выровнялся наоборот, и полоса прокрутки блока сменила своё положение на противоположное. Но при этом сам текст не стал менять своё направление. Это произошло потому, что браузер автоматически задал тексту корректное направление, проанализировав используемые символы Unicode. Чтобы повлиять на это решение браузера существует свойство unicode-bidi, принимающее значения:

* normal — браузер самостоятельно определяет, как ему следует отображать текст на основе используемых символов Unicode;
* embed — переопределяет направление текста, располагая его согласно свойству direction(применяется, когда в блоке текст на двух разнонаправленных языках);
* bidi-override — переопределяет порядок символов в тексте согласно значению direction.
* **Цель 1**Блоку article задайте высоту 200px и overflow: auto,
* **Цель 2**направление текста direction справа налево,
* **Цель 3**а затем параграфу p свойство unicode-bidi: bidi-override

## Направление текста и таблицы [22/26]

Также свойство direction влияет на порядок колонок в таблице, меняя его на соответствующее значение.

## Переполнение текста [23/26]

Ещё одно интересное текстовое свойство — text-overflow. Оно позволяет определить, как будет выглядеть текст, если не поместится в контейнер. Оно принимает разные значения, но универсальными и работающими во всех современных браузерах являются clip и ellipsis.

Значение clip задано по умолчанию, и при нём текст просто обрезается по размеру контейнера, а при ellipsis — обрезается и к концу строки добавляется многоточие.

К сожалению, у этого свойства есть ограничения, которые делают его менее полезным: оно применимо только к однострочным текстам, а также к блокам, значение свойства overflowкоторых установлено в auto, scroll или hidden.

## Интервал между словами [24/26]

И, наконец, последнее свойство, которое мы рассмотрим в этом курсе, — word-spacing. Оно задаёт расстояние между отдельными словами и строчными элементами. Значение может быть положительное или отрицательное и задаётся в px, em, pt или других единицах измерения CSS.

Свойство word-spacing также можно использовать для задания отступов между блочно-строчными элементами, а также изображений, ведь изображения тоже являются строчными.

# Анимация

## Привет, animation!

С помощью CSS можно создавать сложные анимации и очень гибко управлять ими. Описание CSS-анимации состоит из двух частей: набора ключевых кадров keyframesи параметров самой анимации.

Вот пример описания ключевых кадров анимации:

@keyframes stretching {

0% {

width: 100px;

}

100% {

width: 200px;

}

}

Анимация в примере имеет название stretching, и в ней описывается, как будет меняться стиль блока от начальной до конечной точки. Эту анимацию можно применить к любому элементу, для этого достаточно добавить в CSS два свойства — animation-name (название анимации) и animation-duration (длительность) — и задать им нужные значения. Например:

.button {

animation-name: stretching;

animation-duration: 1s;

}

Этот код назначит анимацию stretching элементам с классом button. В результате работы анимации элемент плавно увеличит ширину со 100px до 200px за 1 секунду.

@keyframes rotate {

0% {

transform: rotate(0deg);

}

100% {

transform: rotate(360deg);

}

}

.wood-wheel {

animation-name: rotate;

animation-duration: 15s;

}

## @keyframes: раскадровка

Для каждой анимации нужно задать имя, описать начальный и конечный ключевые кадры, которые задаются с помощью зарезервированных слов from и to или значений 0% и 100%.

Также можно описать промежуточные ключевые кадры, которые задаются с помощью процентов.

Если не задан начальный ключевой кадр, то анимация будет проигрываться из исходного стилевого состояния элемента к ближайшему шагу из перечисленных в keyframes и далее.

Если не задан конечный кадр, то после достижения последнего промежуточного шага, анимация проиграется в обратном направлении до достижения изначального состояния элемента.

Ключевые кадры внутри keyframes могут быть написаны в произвольном порядке, но лучше их перечислять по хронологии от меньшего к большему.

Длительность анимации animation-duration задаётся в секундах или миллисекундах, например: 10s, 100ms.

А теперь давайте попробуем создать анимацию без описания начального кадра, используя шаги в 50% и 100%.

## @keyframes: from и to

Как уже говорилось в [предыдущем задании](https://htmlacademy.ru/courses/80/run/2), начальный и конечный ключевые кадры задаются с помощью слов from и to или значений 0% и 100%.

А промежуточные ключевые кадры задаются с помощью процентов. Вот пример анимации из 4 кадров:

@keyframes coloring {

from { background-color: red; }

33% { background-color: yellow; }

66% { background-color: green; }

to { background-color: blue; }

}

Опробуем from, to и промежуточные кадры в деле!

.platform {

animation-name: lift-up;

animation-duration: 3s;

}

@keyframes lift-up {

from {

transform:translateY(0px);

}

50% { transform:translateY(-250px);}

100% { transform:translateY(-300px);}

}

## @keyframes: группировка кадров

Ключевые кадры в keyframes можно группировать, для этого нужно перечислить их через запятую. Рассмотрим пример:

@keyframes stretching {

0%,

50% {

width: 100px;

}

100% {

width: 200px;

}

}

В этом примере первые два кадра сгруппированы. Анимируемый элемент сначала изменит свою ширину до 100px и останется в этом состоянии половину времени анимации. А за вторую половину времени он растянется от 100px до 200px.

@keyframes lift-up {

0% {

transform: translateY(0px);

}

50%,80% {

transform: translateY(-50px);

}

100% {

transform: translateY(-300px);

}

}

## Множественная анимация, шаг 1

Одному элементу могут быть одновременно назначены несколько анимаций.

Если в этих анимациях меняются разные свойства элемента, то они будут проигрываться одновременно.

В этом и следующем задании попробуем создать множественную анимацию одного объекта. Сначала давайте создадим и назначим первую анимацию.

1. Создайте анимацию rotate с одним ключевым кадром 50%, содержащим трансформацию поворота transform: rotate(360deg).
2. Блоку .stone-wheel задайте свойства animation-name: rotate и animation-duration: 3s

@keyframes rotate {

50% {

transform:rotate(360deg);

}}

.stone-wheel {

animation-name:rotate;

animation-duration:3s;

## Множественная анимация, шаг 2

Теперь разберём, как добавить элементу вторую параллельную анимацию.

Допустим, у нас есть две анимации:

@keyframes move {

to { left: 100px; }

}

@keyframes stretch {

to { width: 100px; }

}

Чтобы назначить элементу вторую анимацию, нужно добавить её название и длительность через запятую в свойствах animation-name и animation-duration. Вот так:

.element {

animation-name: move, stretch;

animation-duration: 5s, 5s;

}

В этом примере две анимации запустятся одновременно, элемент будет параллельно двигаться и удлиняться в течение 5-ти секунд.

Множественные анимации задаются так же, как и множественные фоны и тени — с помощью перечисления свойств через запятую.

Добавим к элементу из [прошлого шага](https://htmlacademy.ru/courses/80/run/5) вторую анимацию.

1. Cоздайте анимацию move с ключевыми кадрами:   
   50% со свойством bottom: 0px;  
   100% со свойством bottom: -50px.
2. Блоку .stone-wheel добавьте вторую анимацию move длительностью 6s

@keyframes rotate {

50% {

transform: rotate(360deg);

}

}

@keyframes move {

50% {

bottom:0px;

}

100% { bottom:-50px;

}

}

.stone-wheel {

animation-name: rotate,move;

animation-duration: 3s,6s;

}

## Водное путешествие

По мере изучения анимаций CSS мы будем наблюдать за развитием цивилизации. И сейчас нам предстоит первое путешествие в новый мир механизмов и шестерёнок.

Ваша задача — создать нужные кадры анимации и отправить корабль в плавание. Свойства animation-name и animation-duration для объектов уже заданы.

@keyframes move-clouds {

to {

transform: translateX(-1000px);

}

}

@keyframes move-sun {

to {

transform: translate(350px, -400px);

}

}

@keyframes move-ship {

to {

transform: translateX(1000px);

}

}

## Количество проигрываний анимации: animation-iteration-count

Во всех предыдущих примерах мы создавали анимации, которые проигрывались один раз, а потом элемент возвращался в исходное состояние. Мы можем определять сколько раз будет повторяться анимация. Для этого используется свойство animation-iteration-count.

В качестве значения оно принимает положительные числа и ноль: при нуле анимация не будет выполнена, в остальных случаях она повторится указанное число раз.

Также в качестве значения animation-iteration-count может быть использовано служебное слово infinite. Оно означает, что анимация будет выполняться бесконечно и никогда не завершится.

Попробуем задавать разное число проигрываний анимации.

## Направление анимации: animation-direction, шаг 1

Помимо количества проигрываний анимации, мы можем определить её направление с помощью свойства animation-direction. По умолчанию анимация имеет прямое направление normal.

Но можно назначить и обратный порядок анимации, чтобы проигрывание начиналось с конца и шло к началу (то есть за начальную точку считался кадр to, а за конечную — from). Для этого используется значение reverse свойства animation-direction.

Попробуем сравнить два направления анимации на примере.

@keyframes clockwise {

to {

transform: rotate(180deg);

}

}

@keyframes anticlockwise {

to {

transform: rotate(-180deg);

}

}

.gear-big {

animation:clockwise;

animation-duration:2s;

animation-direction:reverse;

}

.gear-small {

animation:anticlockwise;

animation-duration:2s;

animation-direction:reverse;

}

## Направление анимации: animation-direction, шаг 2

У свойства animation-direction есть ещё два значения. Они используются, когда количество проигрываний анимации animation-iteration-count больше одного. И оба они определяют чередующееся направление анимации.

Если задано значение alternate, то нечётные проигрывания будут выполняться в прямом направлении, а чётные — в обратном.

.element {

animation-name: move;

animation-duration: 1s;

animation-iteration-count: 2;

animation-direction: alternate;

}

В примере анимация move выполнится два раза: в первый (нечётный) раз направление будет прямым, а во второй (чётный) — обратным.

Если задано значение alternate-reverse, то нечётные проигрывания наоборот будут выполняться в обратном направлении, а чётные — в прямом.

Рассмотрим это на наглядном примере.

.gear-big {

animation-name: clockwise;

animation-duration: 2s;

animation-iteration-count:2;

animation-direction:alternate-reverse;

}

.gear-small {

animation-name: anticlockwise;

animation-duration: 2s;

animation-iteration-count:2;

animation-direction:alternate-reverse;

}

## Задержка начала анимации: animation-delay, шаг 1

Кроме длительности анимации, мы можем управлять задержкой перед началом её выполнения.

В этом и двух следующих заданиях мы создадим две анимации, которые в итоге будут выполняться последовательно с помощью задержки. Сначала давайте создадим и назначим первую анимацию.

1. Создайте анимацию rotate, содержащую кадр to с трансформацией поворота на 360deg,
2. затем добавьте эту анимацию стрелке .arrow-small с длительностью 1s

@keyframes rotate {

to {

transform:rotate(360deg);

}

}

.arrow-small {

animation:rotate;

animation-duration:1s;

}

## Задержка начала анимации: animation-delay, шаг 2

Теперь подготовим вторую анимацию, для которой на следующем шаге зададим задержку.

1. Создайте анимацию ding, содержащую ключевые кадры:   
   33% с трансформацией translateX(-15px);  
   66% с трансформацией translateX(15px).
2. Затем добавьте эту анимацию колокольчику .bell с длительностью 1s

@keyframes ding {

33% {

transform:translateX(-15px);

}

66% {

transform:translateX(15px);

}

}

.bell {

animation:ding;

animation-duration:1s;

## Задержка начала анимации: animation-delay, шаг 3

Синтаксис свойства animation-delay, с помощью которого и назначается задержка начала, идентичен синтаксису свойства animation-duration.

Например, при задании значения animation-delay: 10s анимация начнётся не сразу, а только через десять секунд.

Давайте завершим нашу сцену с часами, чтобы колокольчик звонил только после прохождения стрелкой полного круга.

Колокольчику .bell задайте параметры анимации:

1. задержку начала 1s,
2. длительность 100ms,
3. количество проигрываний анимации 10

.bell {

animation-name: ding;

animation-duration: 100ms;

animation-delay:1s;

animation-iteration-count:10;

}

## Состояние до и после анимации: animation-fill-mode, шаг 1

В предыдущих примерах элементы после проигрывания анимации возвращались в исходное состояние. Но есть свойство, которое определяет, будет ли видимым эффект от анимации, когда сама анимация уже закончилась — это animation-fill-mode. При задании свойству значения forwards элемент будет сохранять состояние после завершения анимации.

Рассмотрим, как работает это свойство, на примере: к двум идентичным объектам применим одинаковую анимацию, а потом у одного изменим значение свойства animation-fill-mode и посмотрим, что будет.

1. Обоим химикатам .reagent задайте анимацию reaction длительностью 3sс задержкой начала в 2s
2. а затем экспериментальному химикату .experimental-reagent задайте сохранение состояния после анимации.

.bubbles {

animation-name: blink;

animation-duration: 2s;

}

@keyframes blink {

50% {

opacity: 1;

}

}

@keyframes reaction {

0% {

background-color: #01ff70;

}

100% {

background-color: #ff4136;

}

}

.reagent {

animation:reaction;

animation-duration:3s;

animation-delay:2s;

}

.experimental-reagent {

animation-fill-mode:forwards;

}

## Состояние до и после анимации: animation-fill-mode, шаг 2

Сохранение свойств анимации по её завершению animation-fill-mode: forwards работает и в случае нескольких повторов анимации или чередующегося направления. Рассмотрим это экспериментально.

Химикату в экспериментальной колбе .experimental-reagent задайте:

1. сохранение своего состояния после анимации,
2. количество проигрываний анимации 2
3. и чередующееся направление анимации.

.experimental-reagent {

animation-fill-mode:forwards;

animation-iteration-count:2;

animation-direction:alternate;

## Состояние до и после анимации: animation-fill-mode, шаг 3

Другое значение свойства animation-fill-mode — backwards. Это значение определяет состояние элемента до начала анимации.

Если элементу назначена анимация с задержкой начала проигрывания и animation-fill-mode: backwards, то стили, описанные в первом ключевом кадре from или 0%, будут применены сразу, ещё до начала проигрывания анимации.

Рассмотрим это на наглядном примере.

1. Заряду аккумуляторов .energy-fill задайте анимацию power длительностью 2sс задержкой начала в 2s
2. а затем заряду в экспериментальном аккумуляторе .mega-energy-fill задайте отображение состояния до начала анимации.

.energy-logo {

animation-name: vibrate;

animation-duration: 2s;

}

@keyframes vibrate {

50% {

transform: scale(1.2);

}

}

@keyframes power {

0% {

width: 150px;

background-color: #01ff70;

}

50% {

background-color: #ffdc00;

}

100% {

width: 30px;

background-color: #ff4136;

}

}

.energy-fill {

animation:power;

animation-duration:2s;

animation-delay:2s;

}

.mega-energy-fill {

animation-fill-mode:backwards;

}

## Состояние до и после анимации: animation-fill-mode, шаг 4

Задание начального состояния анимации до начала её выполнения animation-fill-mode: backwards работает и в случае нескольких повторов или чередующихся направлений анимации. Поставим ещё один эксперимент с аккумуляторами.

## Состояние до и после анимации: animation-fill-mode, шаг 5

Третье значение свойства animation-fill-mode — both.

Оно объединяет действия forwards и backwards. То есть до начала анимации элементу присваивается состояние первого ключевого кадра, а после завершения — конечное состояние анимации сохраняется.

Действие animation-fill-mode: both распространяется и на многоразовую, и на чередующуюся анимацию.

Проведём финальный эксперимент!

.mega-energy-fill {

animation-fill-mode: both;

animation-iteration-count: 2;

animation-direction: alternate;

}

## Остановка и запуск анимации: animation-play-state

Ещё одно управляющее свойство CSS-анимаций — animation-play-state. С его помощью можно поставить анимацию «на паузу», а потом возобновить с места остановки.

Свойство принимает два значения running и paused. Как видно из названий, pausedприостанавливает анимацию, а running начинает или возобновляет анимацию, поставленную на паузу. Значение running задано по умолчанию.

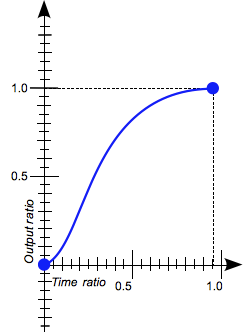
Попробуем создать анимацию и управлять процессом её проигрывания.

## «Форма» анимации, animation-timing-function

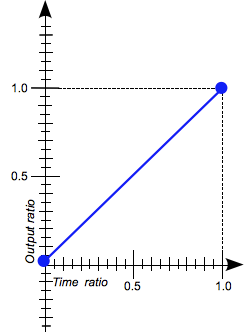
И, наконец, самое интересное свойство — animation-timing-function. Оно определяет, как именно будет происходить анимация: с какой скоростью и ускорением будут меняться свойства, задействованные в ней.

В предыдущих примерах анимация проигрывалась с одинаковой динамикой, мы меняли лишь её длительность, но не «форму». Эта «форма» по умолчанию соответствует первому графику, из которого видно, что анимация начинается медленно, затем ускоряется и к концу движения опять замедляется.

Так ведёт себя значение ease свойства animation-timing-function.

* 

*ease*

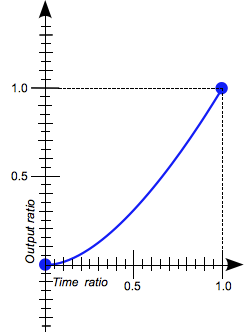
* 

*linear*

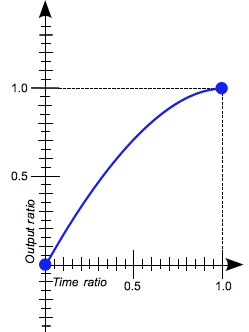
Но мы можем сделать проигрывание анимации равномерным, без ускорений и замедлений. Для этого нужно использовать значение linear. Как видно на втором графике, анимация будет проигрываться с неизменной скоростью.

## animation-timing-function, шаг 2

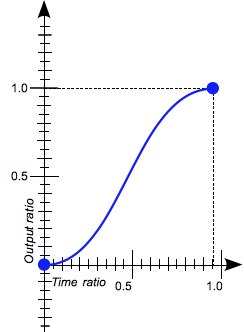
Вот ещё несколько форм анимации: ease-in, ease-out и ease-in-out.

* 

*ease-in*

* 

*ease-out*

* 

*ease-in-out*

Из графиков видно, что при значении ease-in анимация медленно начинается, а к концу ускоряется; при ease-out — начинается быстро, а к концу замедляется. Значение ease-in-outпохоже на ease, то есть анимация начинается и заканчивается медленно, но происходит это чуть-чуть интенсивнее.

Испробуем эти значения свойства animation-timing-function.

## animation-timing-function, шаг 3

Что же скрывается за названиями linear, ease и других функций? Довольно сложная математика кубических [кривых Безье](https://ru.wikipedia.org/wiki/%D0%9A%D1%80%D0%B8%D0%B2%D0%B0%D1%8F_%D0%91%D0%B5%D0%B7%D1%8C%D0%B5).

По сути, именованные функции ease-in, ease-out и другие являются псевдонимами для универсального описания кривых, например:

cubic-bezier(0, 0, 1, 1) // это linear

cubic-bezier(0.42, 0, 1, 1) // это ease-in

В общем представлении cubic-bezier(x1, y1, x2, y2) значения x и y — это координаты точек кривых на графике. При этом верным считается значение x только в диапазоне от 0до 1.

Существует [отличный сервис](http://cubic-bezier.com/), помогающий разобраться в функциональном представлении кривых Безье без необходимости штудировать учебники по математике.

А вот по этой [ссылке](https://easings.net/ru) можно найти целую коллекцию разных easing-функций на основе кривых Безье.

C помощью кривых Безье мы можем задавать любые формы анимации. Давайте попробуем!

## animation-timing-function, шаг 4

Давайте разберёмся с ещё одним возможным классом значений animation-timing-function — это steps.

Они позволяют задать «ступеньки», по которым будет идти анимация. Синтаксис stepsследующий:

animation-timing-function: steps(число\_шагов, направление);

Тут всё просто: число шагов — это целое число, за которое будет выполнена вся анимация; направление может принимать значение start или end.

При заданном start первый шаг выполняется одновременно с началом анимации, а в случае c end последний шаг будет выполнен вместе с завершением анимации. То есть при start пошаговая анимация идёт как бы с опережением, а при end — вдогонку.

Разберём это наглядно.

1. Роботу .constructor задайте форму анимации steps(5, start)
2. а затем измените её на steps(5, end)

## Посадка, шаг 3

<!DOCTYPE html>

<html lang="ru">

<head>

<title>Посадка, шаг 3</title>

<meta charset="utf-8">

<link rel="stylesheet" href="epoch5.css">

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

</head>

<body class="cosmic">

<div class="moon">

<span class="flag"></span>

</div>

<div class="rocket arrival">

<span class="fuel"></span>

</div>

</body>

</html>

.rocket {

animation-name: fly;

animation-duration: 1.5s;

animation-timing-function: ease-out;

animation-fill-mode: forwards;

}

.fuel {

animation-name: fire;

animation-duration: 1s;

animation-delay: 1s;

animation-fill-mode: forwards;

}

@keyframes fly {

to {

transform: translate(240px, 260px) rotate(-45deg);

}

}

@keyframes fire {

to {

opacity: 0;

}

}

@keyframes hoist {

to {

top:-25px;

}

}

.flag {

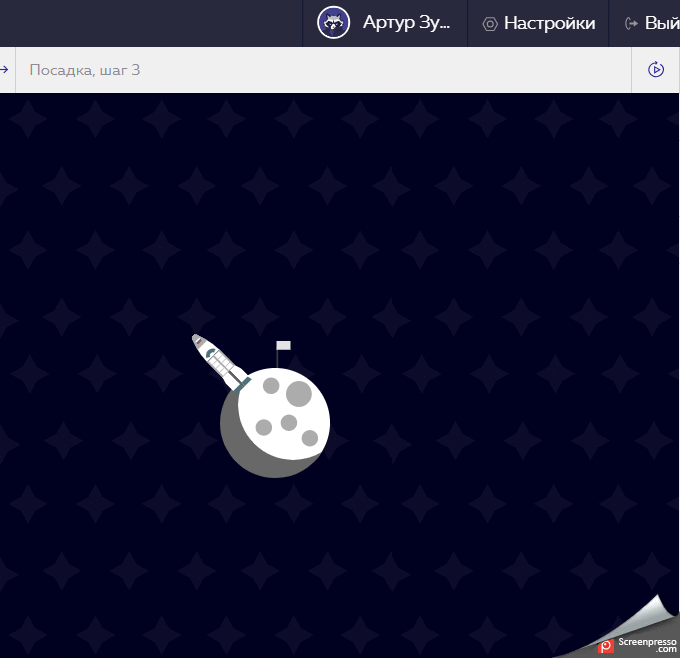
animation:hoist;

animation-duration:1s;

animation-delay:2s;

animation-timing-function:ease-in;

animation-fill-mode:forwards;

}

# Плавные переходы

## Длительность перехода, transition-duration [1/29]

Плавные переходы в CSS или transitions позволяют изменять значения свойств постепенно, «с нежностью».

Обычно эффект от изменения значений CSS-свойств виден мгновенно, но с помощью плавных переходов можно изменить это поведение и сделать процесс изменения значений достаточно длительным.

В отличие от [анимаций](https://htmlacademy.ru/courses/80), которые позволяют управлять любым количеством промежуточных состояний, с помощью transitions можно управлять только переходом между двумя состояниями: начальным и конечным.

Чтобы обозначить плавный переход в CSS, достаточно задать одно свойство: transition-duration — длительность перехода. Значения задаются в секундах (10s, 3s), долях секунды (0.1s, 0.03s) или миллисекундах (100ms, 333ms).

Рассмотрим плавный переход на простом примере с кнопкой, у которой по нажатию меняется цвет. Начальное состояние описано в классе .btn, а конечное — в классе .btn-active.

Класс btn-active в этом и следующих заданиях добавляется кнопке при нажатии с помощью JavaScript.

.btn {

background-color: #ffffff;

transition-duration:1s;

}

.btn-active {

background-color: #ff5252;

}

Курс «[Плавные переходы](https://htmlacademy.ru/courses/84)»

## Длительность перехода, шаг 2 [2/29]

Плавные переходы и анимацию можно применить только к некоторым CSS-свойствам. В основном это свойства, изменяющие размер, цвет, позицию элементов. В [статье на MDN](https://developer.mozilla.org/en-US/docs/Web/CSS/CSS_animated_properties)приведен перечень анимируемых свойств и характер их анимирования.

Если задана только длительность перехода transition-duration, то по умолчанию плавное изменение затрагивает все свойства элемента и для всех свойств происходит одновременно.

Давайте попробуем задать кнопке ещё несколько свойств и посмотрим, как они будут изменяться.

В этом и следующих примерах используются CSS-трансформации, которые подробно разбираются в [курсе «Двумерные трансформации»](https://htmlacademy.ru/courses/71).

## transition-property: какие свойства изменять плавно? [3/29]

По умолчанию плавный переход действует на все анимируемые свойства элемента. Такое поведение соответствует значению all свойства transition-property, это значение по умолчанию.

Такое поведение часто нежелательно, особенно когда в правиле много свойств.

Можно указать, какие именно свойства нужно изменять плавно, перечислив их в transition-property через запятую:

transition-property: width; // плавно меняется только ширина

transition-property: width, height; // плавно меняются только ширина и высота

При этом так же через запятую можно задавать переходам разных свойств разную длительность:

transition-property: width, height;

transition-duration: 1s, 5s; // ширина меняется за 1 секунду, высота за 5

Попробуем применить описанное свойство на практике. В этом примере у кнопки анимируются свойства background-color, transform и font-size.

.btn {

color: #ffffff;

background-color: #00e676;

transition-duration: 1s,3s;

transition-property:background-color, transform;

}

.btn-active {

font-size: 60px;

background-color: #ff5252;

transform: rotate(180deg);

}

## Задержка перехода, transition-delay [4/29]

Ещё одно свойство переходов — transition-delay. Оно определяет задержку перед началом перехода и задаётся в секундах или миллисекундах, как и transition-duration.

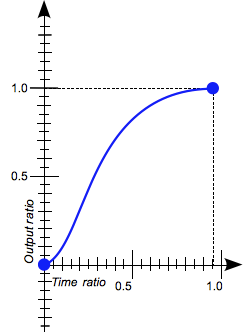
С ним всё просто. Разберёмся на примере.

## «Форма» перехода, transition-timing-function [5/29]

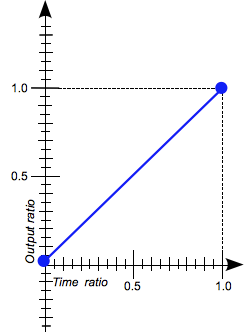
Ещё одно свойство, влияющее на переход — transition-timing-function. Это свойство аналогично свойству animation-timing-function, которое разбирается в [курсе по анимациям](https://htmlacademy.ru/courses/80/run/23). Свойство transition-timing-function определяет с какой скоростью и ускорением будут меняться свойства во время перехода.

В предыдущих примерах переходы происходили с одинаковой динамикой. Мы меняли длительность и задержку перехода, но не «форму». Эта «форма» по умолчанию соответствует первому графику, из которого видно, что переход начинается медленно, затем ускоряется и к концу движения опять замедляется.

Так ведёт себя значение ease свойства transition-timing-function.

* 

*ease*

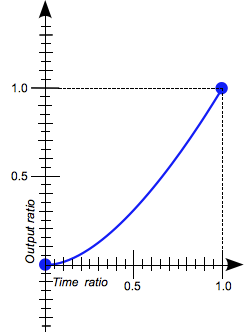
* 

*linear*

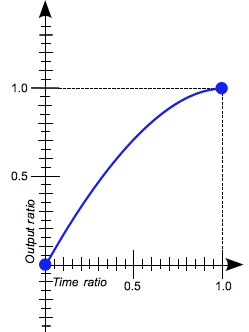
Чтобы сделать переход равномерным, без ускорений и замедлений, используется значение linear. «Форма» линейного перехода приведена на втором графике.

## transition-timing-function, шаг 2 [6/29]

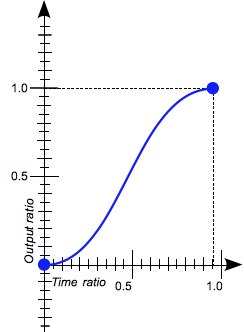
Вот ещё несколько форм переходов: ease-in, ease-out и ease-in-out.

* 

*ease-in*

* 

*ease-out*

* 

*ease-in-out*

Из графиков видно, что при значении ease-in переход медленно начинается, а к концу ускоряется; при ease-out — начинается быстро, а к концу замедляется. Значение ease-in-outпохоже на ease, то есть переход начинается и заканчивается медленно, но происходит это чуть-чуть интенсивнее.

Испробуем эти значения свойства transition-timing-function.

## transition-timing-function, шаг 3 [7/29]

Названия linear, ease, ease-in и другие — это «псевдонимы» функций кубических [кривых Безье](https://ru.wikipedia.org/wiki/%D0%9A%D1%80%D0%B8%D0%B2%D0%B0%D1%8F_%D0%91%D0%B5%D0%B7%D1%8C%D0%B5):

cubic-bezier(0, 0, 1, 1) // это linear

cubic-bezier(0.42, 0, 1, 1) // это ease

В общем представлении cubic-bezier(x1, y1, x2, y2) значения x и y — это координаты точек кривых на графике. При этом верным считается значение x только в диапазоне от 0 до 1.

Существует [отличный сервис](http://cubic-bezier.com/), помогающий разобраться в функциональном представлении кривых Безье без необходимости штудировать учебники по математике.

А вот по этой [ссылке](http://easings.net/ru) можно найти целую коллекцию разных easing-функций на основе кривых Безье.

C помощью функции cubic-bezier мы можем задавать любые формы переходов.

## transition-timing-function, шаг 4 [8/29]

Ещё один возможный класс значений transition-timing-function — это steps.

Они позволяют задать «ступеньки», по которым будет идти переход. Синтаксис stepsследующий:

transition-timing-function: steps(число\_шагов, направление);

Тут всё просто: число\_шагов — это целое число, за которое будет выполнен переход; направление может принимать значение start или end.

При заданном start первый шаг выполняется одновременно с началом перехода, а в случае c end последний шаг будет выполнен вместе с завершением перехода.

Кстати, переход можно описать в сокращенном виде свойством transition. Параметры перехода просто перечисляются через пробел: свойство, длительность, форма и задержка:

transition: width 1s ease-in 2s;

К примеру, переход тут применяется к ширине элемента, будет длиться 1 секунду с формой ease-in и задержкой перед началом в 2 секунды.

Описание перехода для квадрата .square в этом задании мы тоже перевели в короткую форму.

## Бумажные кнопки, шаг 1 [9/29]

Как вы уже наверное заметили, курс выполнен в общей стилистике Material Design — спецификации дизайна от Google.

Давайте воспользуемся полученными знаниями о переходах и попробуем изобрести заново базовые элементы формы: текстовые поля, чекбоксы, переключатели и другие. В нашей реализации они будут следовать принципам Material Design и использовать плавные переходы при смене состояний.

Начнём с «бумажных» кнопок с тенями. Реализуем их двумя парами радио-кнопок и подписей к ним. Создадим радио-кнопки и подписи, а затем визуально скроем радио-кнопки.

1. **Цель 1**В блоке .paper-btn-true раскоментируйте радио-кнопку и подпись к ней, в подпись добавьте текст Правда,
2. **Цель 2**затем раскомментируйте радио-кнопку и подпись в блоке .paper-btn-false, в подпись добавьте текст Ложь.
3. **Цель 3**Скройте поля ввода внутри .paper-btn с помощью display: none.

.paper-btn {

display: inline-block;

margin: 58px 12px;

}

.paper-btn label {

display: inline-block;

width: 80px;

line-height: 40px;

font-size: 14px;

text-transform: uppercase;

color: #616161;

background-color: #f5f5f5;

border-radius: 3px;

cursor: pointer;

box-shadow: 0 2px 5px rgba(0, 0, 0, 0.3);

}

.paper-btn input {

display:none;

}

## Бумажные кнопки, шаг 2 [10/29]

Итак, у нас есть заготовки кнопок. Давайте оживим их.

Переключаться кнопки будут при помощи трюка с селекторами ~ и :checked, который описан в [задании курса «Селекторы. Часть 3».](https://htmlacademy.ru/courses/58/run/17)

Сначала зададим цвет фона для активного состояния кнопок. Кнопка «Правда» будет менять цвет на зелёный, а кнопка «Ложь» — на красный. Для этого нам нужно будет воспользоваться такими селекторами:

.paper-btn-true input:checked ~ label { ... } // правда

.paper-btn-false input:checked ~ label { ... } // ложь

1. **Цель 1**Для активного состояния кнопки «Правда» задайте цвет фона #00e676,
2. **Цель 2**а для активного состояния кнопки «Ложь» — цвет фона #ff5252.

Переключайте кнопки «Правда» или «Ложь» для проверки результата.

.paper-btn input {

display: none;

}

.paper-btn-true input:checked ~ label {

background-color:#00e676;

}

.paper-btn-false input:checked ~ label {

Backgro

## Бумажные кнопки, шаг 3 [11/29]

Отлично! Теперь добавим к активному состоянию кнопок более контрастную тень и изменим цвет текста. Для этого создадим CSS-правило с таким селектором:

.paper-btn input:checked ~ label { ... }

Тень box-shadow, которая появляется в активном состоянии, анимируем с помощью плавного перехода. Cвойства перехода нужно добавить в общее CSS-правило для подписи:

.paper-btn label { ... }

Так тень будет и появляться, и исчезать плавно.

1. **Цель 1**Для активного состояния кнопок задайте цвет текста #f1f1f1 и тень 0 6px 20px 0 rgba(0, 0, 0, 0.4).
2. **Цель 2**А затем обеим кнопкам задайте плавный переход для свойства box-shadow длительностью 0.3s с формой cubic-bezier(0.4, 0, 0.2, 1).

.paper-btn {

display: inline-block;

margin: 58px 12px;

}

.paper-btn label {

display: inline-block;

width: 80px;

line-height: 40px;

font-size: 14px;

text-transform: uppercase;

color: #616161;

background-color: #f5f5f5;

border-radius: 3px;

cursor: pointer;

box-shadow: 0 2px 5px rgba(0, 0, 0, 0.3);

transition:box-shadow 0.3s cubic-bezier(0.4, 0, 0.2, 1) 0s;

}

.paper-btn input {

display: none;

}

.paper-btn input:checked ~ label {

color:#f1f1f1;

box-shadow:0 6px 20px 0 rgba(0, 0, 0, 0.4);

}

.paper-btn-true input:checked ~ label {

background-color: #00e676;

}

.paper-btn-false input:checked ~ label {

background-color: #ff5252;

}

## Чекбоксы, шаг 1 [12/29]

Теперь давайте займёмся усовершенствованием чекбоксов. Воспользуемся приёмом с :checked ~ из [задания курса про селекторы](https://htmlacademy.ru/courses/58/run/17) снова.

Для начала добавим в разметку чекбоксы и подписи к ним, а потом спрячем чекбоксы.

Затем с помощью псевдоэлемента ::before у подписей сымитируем чекбосы с нужным внешним видом: маленькие квадраты с тёмной границей. Стили для псевдоэлементов уже заданы в коде.

1. **Цель 1**Раскомментируйте чекбоксы и подписи к ним.
2. **Цель 2**В первую подпись добавьте текст absolute,
3. **Цель 3**во вторую — текст great,
4. **Цель 4**в третью — текст majestic.
5. **Цель 5**Спрячьте чекбоксы с помощью display: none

label {

position: relative;

cursor: pointer;

}

label::before {

content: "";

position: absolute;

top: 0;

left: -44px;

z-index: 1;

width: 16px;

height: 16px;

border: 2px solid #5a5a5a;

}

input[type="checkbox"] {

display:none;

}

## Чекбоксы, шаг 2 [13/29]

Поработаем над выделенным состоянием нашего label::before или «псевдочекбокса».

Псевдочекбокс при выделении давайте преобразуем в «галочку» — сплющенный блок бирюзового цвета без верхней и правой рамок.

Для управления выделенным состоянием псевдочекбокса создадим CSS-правило с таким селектором:

input[type="checkbox"]:checked ~ label::before { ... }

Для выделенного состояния «псевдочекбокса» задайте:

1. **Цель 1**ширину 21px и высоту 10px,
2. **Цель 2**цвет рамки #00bad2,
3. **Цель 3**стиль верхней и правой рамки none.

input[type="checkbox"]:checked ~ label::before {

width:21px;

height:10px;

border-color:#00bad2;

border-top:none;

border-right:none;

}

## Чекбоксы, шаг 3 [14/29]

А теперь последний штрих: будем плавно поворачивать псевдочекбокс в выделенном состоянии на 45 градусов. Чтобы он стал похож на «галочку».

В правило для выделенного состояния:

input[type="checkbox"]:checked ~ label::before { ... }

мы добавим трансформацию поворота, а свойства для описания плавного перехода добавим в правило:

label::before { ... }

Это нужно для того, чтобы поворот происходил плавно в обе стороны: при выделении и при снятии выделения.

Да, к псевдоэлементам тоже можно запросто применять переходы, как и к обычным элементам!

Далее мы будем интенсивнее использовать трансформации, поэтому вы можете повторить их в [курсе «Двумерные трансформации»](https://htmlacademy.ru/courses/71).

1. **Цель 1**Для выделенного состояния «псевдочекбокса» задайте поворот transform: rotate(-45deg).
2. **Цель 2**А затем для label::before задайте плавный переход для всех свойств длительностью 0.3s с формой ease-in-out.

Переключайте чекбоксы для проверки результата.

label {

position: relative;

cursor: pointer;

}

label::before {

content: "";

position: absolute;

top: 0;

left: -44px;

z-index: 1;

width: 16px;

height: 16px;

border: 2px solid #5a5a5a;

transition:0.3s ease-in-out;

}

input[type="checkbox"] {

display: none;

}

input[type="checkbox"]:checked ~ label::before {

width: 21px;

height: 10px;

border-color: #00bad2;

border-top-style: none;

border-right-style: none;

transform:rotate(-45deg);

}

## Радио-кнопки, шаг 1 [15/29]

Давайте теперь займёмся радио-кнопками. С помощью переходов мы можем анимировать момент их выделения.

Тут мы снова воспользуемся приёмом :checked ~, а также будем задавать переходы для псевдоэлементов.

Сначала создадим радио-кнопки и подписи к ним, затем скроем радио-кнопки, а подписи оставим. В качестве «кружков» радио-кнопок будут выступать псевдоэлементы подписей label::before. Для них стили уже готовы.

1. **Цель 1**Раскомментируйте радио-кнопки и подписи к ним.
2. **Цель 2**В первую подпись добавьте текст transparency,
3. **Цель 3**во вторую — текст opacity,
4. **Цель 4**в третью — текст invisibility.
5. **Цель 5**Спрячьте радио-кнопки с помощью display: none

label {

position: relative;

cursor: pointer;

}

label::before,

label::after {

content: "";

position: absolute;

top: 0;

left: -42px;

border-radius: 50%;

}

label::before {

width: 12px;

height: 12px;

border: solid 2px #5a5a5a;

}

input[type="radio"] {

display:none;

}

## Радио-кнопки, шаг 2 [16/29]

Основа радио-кнопок готова. Теперь давайте разберёмся с их выделением.

В качестве «выделяющего элемента» будем использовать псевдоэлемент подписи label::after.

Зададим label::after ширину и высоту, цвет и немного увеличим трансформацией.

1. **Цель 1**Для label::after задайте ширину и высоту 16px,
2. **Цель 2**цвет фона #00bad2,
3. **Цель 3**трансформацию transform: scale(1.1).

## Радио-кнопки, шаг 3 [17/29]

Теперь оживим радио-кнопки.

Для начала уменьшим закрывающий псевдоэлемент label::after или «закрывашку» до нулевого размера с помощью трансформации scale.

А в активном состоянии будем увеличивать «закрывашку» до нужного размера. Для управления активным состоянием «закрывашки» используем CSS-правило с таким селектором:

input[type="radio"]:checked ~ label::after { ... }

Свойства плавного перехода, как и в предыдущих примерах, добавим в CSS-правило для исходного состояния:

label::after { ... }

1. **Цель 1**В CSS-правиле label::after измените трансформацию на transform: scale(0).
2. **Цель 2**Для активного состояния «закрывашки» задайте трансформацию transform: scale(1.1).
3. **Цель 3**Наконец в правиле label::after задайте плавный переход для свойства transform длительностью 0.28s.

Переключайте радио-кнопки для проверки результата.

label {

position: relative;

cursor: pointer;

}

label::before,

label::after {

content: "";

position: absolute;

top: 0;

left: -42px;

border-radius: 50%;

}

label::before {

width: 12px;

height: 12px;

border: solid 2px #5a5a5a;

}

label::after {

width: 16px;

height: 16px;

background-color: #00bad2;

transform: scale(0);

transition:transform 0.28s;

}

input[type="radio"] {

display: none;

}

input[type="radio"]:checked ~ label::after {

transform:scale(1.1);

}

## Переключатели, шаг 1 [18/29]

У ещё одного элемента формы, который мы соберём, нет «нативного» браузерного аналога — это переключатель.

Переключатели больше распространены в мобильных интерфейсах.

Для этого нам понадобятся уже полюбившиеся вам чекбоксы, подписи и приём с :checked ~. Давайте как обычно создадим в разметке нужные элементы и скроем «нативные» чекбоксы.

1. **Цель 1**Раскомментируйте чекбоксы и подписи к ним.
2. **Цель 2**В первую подпись добавьте текст nextid,
3. **Цель 3**во вторую — текст bgsound,
4. **Цель 4**в третью — текст hole.
5. **Цель 5**Спрячьте чекбоксы с помощью display: none.

## Переключатели, шаг 2 [19/29]

Используем псевдоэлемент label::before для построения «шкалы» переключателя: зададим ему размеры, цвет рамки и положение в пространстве.

1. **Цель 1**Для label::before задайте ширину 32px и высоту 1px,
2. **Цель 2**сплошную верхнюю рамку с толщиной 1px и цветом #5a5a5a,
3. **Цель 3**а затем координаты top: 7px и left: 108px

label {

position: relative;

display: inline-block;

max-width: 100px;

word-wrap: break-word;

cursor: pointer;

}

label::before,

label::after {

content: "";

position: absolute;

display: block;

}

label::before {

width:32px;

height:1px;

border-top:1px solid #5a5a5a;

top:7px;

left:108px;

}

input[type="checkbox"] {

display: none;

}

## Переключатели, шаг 3 [20/29]

Теперь из псевдоэлементов label::after сделаем ползунки переключателей.

Зададим ползункам размеры, рамки, цвет, положение на экране и скруглим им края.

1. **Цель 1**Для label::after задайте ширину и высоту 12px,
2. **Цель 2**сплошную рамку с толщиной 2px и цветом #5a5a5a, скругление углов border-radius: 50%,
3. **Цель 3**координаты top: 0 и left: 100px,
4. **Цель 4**а затем цвет фона #ffffff.

## Переключатели, шаг 4 [21/29]

Осталось научить ползунки, которые мы сделали с помощью псевдоэлемента label::after, переключаться.

Для управления активным состоянием ползунка используем CSS-правило с таким селектором:

input[type="checkbox"]:checked ~ label::after { ... }

В активном состоянии мы будем смещать ползунок с помощью трансформации translateX, а также менять его цвет.

Свойства плавного перехода, как и в предыдущих примерах, добавим в CSS-правило для исходного состояния ползунка:

label::after { ... }

В этом случае он будет плавно перемещаться в обоих направлениях.

1. **Цель 1**Для активного состояния ползунка задайте трансформацию перемещения translateX(32px),
2. **Цель 2**цвет фона и цвет рамок #0f9d58.
3. **Цель 3**А затем в правиле label::after задайте переход для всех свойств длительностью 200ms с формой ease-in.

Нажимайте на переключатели для проверки результата.

label {

position: relative;

display: inline-block;

max-width: 100px;

word-wrap: break-word;

cursor: pointer;

}

label::before,

label::after {

content: "";

position: absolute;

display: block;

}

label::before {

top: 7px;

left: 108px;

width: 32px;

height: 1px;

border-top: 1px solid #5a5a5a;

}

label::after {

top: 0px;

left: 100px;

width: 12px;

height: 12px;

background-color: #ffffff;

border: solid 2px #5a5a5a;

border-radius: 50%;

transition:200ms ease-in;

}

input[type="checkbox"] {

display: none;

}

input[type="checkbox"]:checked ~ label::after {

transform:TranslateX(32px);

border-color:#0f9d58;

background-color:#0f9d58;

}

## Иконка-трансформер, шаг 1 [22/29]

Давайте немного отдохнём от элементов форм и создадим трансформирующуюся иконку.

«Иконка-гамбургер» — привычный для мобильных приложений и сайтов глиф, обозначающий меню. Обычно по клику на него появляется меню или меняется текущее отображение экрана. И, чтобы вернуться к прежнему отображению, нужно снова нажать на иконку.

С помощью трансформаций и переходов можно сделать такую иконку интерактивной: она своим видом будет подсказывать, что повторный клик на неё вернёт прошлое отображение.

Итак, начнём! Воспользуемся псевдоэлементами .menu-icon span::before и .menu-icon span::after для создания «иконки-гамбургера».

В этом и следующих примерах используются трансформации CSS, которые подробно разбираются в [курсе «Двумерные трансформации»](https://htmlacademy.ru/courses/71).

1. **Цель 1**Для .menu-icon span задайте высоту 12px и цвет фона #40d47e.
2. **Цель 2**Затем задайте такую же высоту и фоновый цвет для span::before и span::after.
3. **Цель 3**Потом только span::before задайте трансформацию translate(0, -24px),
4. **Цель 4**а span::after — трансформацию translate(0, 12px).

## Иконка-трансформер, шаг 2 [23/29]

Гамбургер готов! Теперь давайте поработаем над его открытым состоянием, в котором он будет превращаться в стрелку.

В открытом состоянии мы будем поворачивать иконку, при этом верхнюю и нижнюю полоску немного укоротим, чтобы на следующем шаге превратить их в «указатель» стрелки.

Для перевода в открытое состояние будем добавлять к иконке класс menu-icon-open. Конечно, переключение класса будет сделано с помощью JavaScript по щелчку. Но чтобы процесс создания стрелки был более наглядным, мы добавили в HTML-код вторую иконку уже в открытом состоянии.

1. **Цель 1**Для .menu-icon-open span::before и .menu-icon-open span::after задайте ширину 60%.
2. **Цель 2**Для .menu-icon-open span задайте трансформацию rotate(180deg).

<!DOCTYPE html>

<html lang="ru">

<head>

<title>Иконка-трансформер, шаг 2</title>

<meta charset="utf-8">

<base href="/assets/course84/">

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

</head>

<body class="blue-theme">

<section class="card card-bordered">

<div class="menu-icon">

<span></span>

</div>

<div class="menu-icon menu-icon-open">

<span></span>

</div>

</section>

</body>

</html>

.menu-icon span {

display: block;

width: 100%;

height: 12px;

background-color: #40d47e;

}

.menu-icon span::before,

.menu-icon span::after {

content: "";

display: block;

width: 100%;

height: 12px;

background-color: #40d47e;

}

.menu-icon span::before {

transform: translate(0, -24px);

}

.menu-icon span::after {

transform: translate(0, 12px);

}

.menu-icon-open span {

transform:rotate(180deg);

}

.menu-icon-open span::before,

.menu-icon-open span::after {

width:60%;

}

## Иконка-трансформер, шаг 3 [24/29]

Отлично! Теперь с помощью трансформаций сместим и повернём короткие прямоугольники так, чтобы получилась стрелка.

Напоминаем, что к элементам или псевдоэлементам можно применить несколько трансформаций одновременно, для этого их нужно перечислить через пробел, например так:

.some-class::before {

transform: translate(15px, 15px) rotate(90deg);

}

Порядок следования трансформации важен, так как при разном порядке могут получаться разные результаты.

1. **Цель 1**Для .menu-icon-open span::before задайте трансформацию translate(45px, -14px),
2. **Цель 2**а затем дополнительно поверните rotate(45deg).
3. **Цель 3**Для .menu-icon-open span::after задайте трансформацию translate(45px, 2px),
4. **Цель 4**а затем дополнительно поверните rotate(-45deg).

.menu-icon-open span::before {

transform:translate(45px,-14px) rotate(45deg);

}

.menu-icon-open span::after {

transform:translate(45px,2px) rotate(-45deg);

}

## Иконка-трансформер, шаг 4 [25/29]

И, наконец, добавим для элементов иконки переходы, чтобы превращение «гамбургера» в стрелку проходило плавно.

Щёлкайте на иконку и смотрите, какой крутой эффект получился.

Задайте плавный переход для всех свойств длительностью 0.7s с формой ease-in-out для:

1. **Цель 1**.menu-icon span,
2. **Цель 2**.menu-icon span::before и .menu-icon span::after.

Нажимайте на иконку, чтобы переключать класс menu-icon-open для неё.

.menu-icon span {

display: block;

width: 100%;

height: 12px;

background-color: #40d47e;

transition:0.7s ease-in-out;

}

.menu-icon span::before,

.menu-icon span::after {

content: "";

display: block;

width: 100%;

height: 12px;

background-color: #40d47e;

transition:0.7s ease-in-out;

}

.menu-icon span::before {

transform: translate(0, -24px);

}

.menu-icon span::after {

transform: translate(0, 12px);

}

.menu-icon-open span {

transform: rotate(180deg);

}

.menu-icon-open span::before,

.menu-icon-open span::after {

width: 60%;

}

.menu-icon-open span::before {

transform: translate(45px, -14px) rotate(45deg);

}

.menu-icon-open span::after {

transform: translate(45px, 2px) rotate(-45deg);

}



## Текстовое поле ввода, шаг 1 [26/29]

Вернёмся к формам и последним в этом курсе усовершенствуем «нативное» текстовое поле ввода.

Для начала создадим само поле ввода и подпись к нему, которая станет в дальнейшем интерактивным плейсхолдером. И начнём стилизовать элементы.

1. **Цель 1**Раскомментируйте текстовое поле ввода и подпись к нему.
2. **Цель 2**У поля ввода уберите все рамки с помощью border: none и задайте сплошную нижнюю рамку толщиной 1px и цветом #757575.
3. **Цель 3**Для label задайте абсолютное позиционирование и координату top: -36px.

<!DOCTYPE html>

<html lang="ru">

<head>

<title>Текстовое поле ввода, шаг 1</title>

<meta charset="utf-8">

<base href="/assets/course84/">

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

</head>

<body class="purple-theme">

<section class="card">

<h1 class="question"><span>Как назывался самый первый браузер?</span></h1>

<form class="answers">

<button class="fab">&#58882;</button>

<div>

<input type="text" required>

<div class="label-box">

<label>Слово по-английски</label>

</div>

</div>

</form>

</section>

</body>

</html>

.label-box {

position: relative;

width: 270px;

}

input {

width: 270px;

padding: 5px;

font-size: 18px;

box-sizing: border-box;

border:none;

border-bottom:1px solid #757575;

}

label {

font-size: 18px;

color: #757575;

pointer-events: none;

position:absolute;

top:-36px;

}

## Текстовое поле ввода, шаг 2 [27/29]

В предыдущих заданиях много раз использовался приём с :checked ~. В этом задании воспользуемся похожим приёмом для селекторов :focus и :valid, которые тоже разбираются в курсе [«Селекторы. Часть 3»](https://htmlacademy.ru/courses/58).

Чтобы наш самодельный плейсхолдер не мешал при вводе текста и не закрывал заполненные поля, сместим его выше и немного уменьшим. Для этого используем CSS-правило с такими селекторами:

input:focus ~ .label-box label,

input:valid ~ .label-box label {

...

}

Первый селектор срабатывает когда в поле находится курсор, а второй селектор когда поле заполнено верно. Это упрощённый пример, поэтому мы не учитываем все варианты, например, когда поле заполнено неверно.

Чтобы подпись смещалась красиво, добавим ей плавный переход.

1. **Цель 1**При сфокусированном или валидном состоянии поля ввода задайте для label размер шрифта 14px и трансформацию translateY(-20px).
2. **Цель 2**Затем в общее CSS-правило для label добавьте переход для всех свойств длительностью 0.2s.

## Текстовое поле ввода, шаг 3 [28/29]

еперь, когда плейсхолдер готов, сделаем декоративное выделение сфокусированного поля ввода с помощью псевдоэлементов у .label-box.

Это выделение будет выглядеть, как дополнительная нижняя рамка у поля ввода. Рамка будет появляться из центра поля, когда в нём появляется курсор, и исчезать при потере полем фокуса.

Создадим декоративную рамку из двух кусочков. Используем .label-box::before и .label-box::after для каждой части.

1. **Цель 1**Псевдоэлементу .label-box::before задайте: содержимое content: "", абсолютное позиционирование, высоту 2px, ширину 50% и фоновый цвет #757575,
2. **Цель 2**а затем координаты bottom: 1px и left: 50%.
3. **Цель 3**Псевдоэлементу .label-box::after задайте: содержимое content: "", абсолютное позиционирование, высоту 2px, ширину 50% и фоновый цвет #757575,
4. **Цель 4**а затем координаты bottom: 1px и right: 50%.

## Текстовое поле ввода, шаг 4 [29/29]

Осталось оживить декоративное выделение.

В исходном состоянии выделение отсутствует, поэтому сначала изменим его ширину на нулевую. Чтобы декоративное выделение появлялось, когда поле ввода получает фокус, используем CSS-правило с такими селекторами:

input:focus ~ .label-box::before,

input:valid ~ .label-box::after {

...

}

И в этом правиле зададим нужную ширину частей рамки.

Чтобы выделение появлялось и исчезало плавно, нужно для обоих его частей задать плавный переход в этом правиле:

.label-box::before,

.label-box::after {

...

}