# Конспект «Основы HTML и CSS»

## HTML

HTML расшифровывается как «Hypertext Markup Language», то есть «язык гипертекстовой разметки».

Язык HTML отвечает за структуру и содержание страницы. HTML состоит из тегов, а тег состоит из имени, заключённого между знаками «меньше» и «больше». Примеры тегов: <h1>, <p>, <ul>.

### **Парные теги**

Теги бывают парными и одиночными. Парные теги могут содержать текст и другие теги. У парных тегов, в отличие от одиночных, есть вторая половинка — закрывающий тег:

<h1>Текст заголовка</h1>

В закрывающей части парных тегов перед именем ставится символ / («слэш»).

В парные теги можно вкладывать другие теги. Например, как в списках:

<ul>

<li>Элемент списка</li>

</ul>

У вложенных тегов всегда нужно следить за правильным порядком закрытия. Вложенный тег не может закрываться позже родительского:

<ul><li>Элемент списка</ul></li> <!-- Плохо -->

<ul><li>Элемент списка</li></ul> <!-- Хорошо -->

Не все теги можно вкладывать в другие теги, например, тег <h1> нельзя вкладывать в <p>.

### **Одиночные теги**

Существуют не только парные, но и одиночные теги. Например, тег <img> позволяет добавить картинку в разметку.

Сам по себе <img> не имеет смысла. Чтобы этот тег был действительно полезен, необходимо написать внутри него адрес, ведущий к картинке. Делается это с помощью атрибута src:

<img src="keks.png">

У тега может быть несколько атрибутов. В этом случае они пишутся через пробел:

<тег атрибут1="значение1" атрибут2="значение2">

Например, картинке при желании можно задать размеры:

<img src="keks.png" width="200" height="100">

### **Комментарии**

Код, заключённый между символами <!-- и -->, работать не будет. Если эти символы удалить, то код заработает, это называется «раскомментировать». С помощью комментариев обычно временно отключают какой-то код или оставляют подсказки и разъяснения.

<!-- Это комментарий в HTML -->

## CSS

CSS расшифровывается как «Cascading Style Sheets», то есть «каскадные таблицы стилей».

Язык CSS отвечает за внешний вид страницы.

С помощью CSS можно задавать параметры для любого тега: ширину и высоту, отступы, цвет и размер шрифта, фон и так далее. Все эти параметры задаются с помощью свойств в следующем формате:

свойство: значение;

Например:

color: red;

padding: 10px;

Стили к тегам добавляются чаще всего при помощи атрибута class.

Например, если мы хотим, чтобы определённые стили, описанные, допустим, в классе feature-kitten, применились к тегу <p>, то в разметке напишем так:

<p class="feature-kitten">...</p>

### **CSS-правила**

Можно сказать, что CSS-правило — это группа свойств и их значений, которая целиком применяется к тем тегам, на которые указывает селектор.

И выглядит это так:

селектор {

свойство1: значение;

свойство2: значение;

}

Задавать стили можно не только с помощью атрибута class, но и по тегам. Селектор указывает, к каким тегам применятся свойства из CSS-правила. Селекторы по тегам работают проще всего: они выбирают все теги с подходящим именем.

p { color: red; }

В примере селектором является p, и он выбирает все теги с именем p (то есть теги <p>), а теги с другим именем, например h1, не выбирает.

Когда же стилизация задаётся по классам, то стили применяются только к тегам с такими классами.

.название\_класса {

свойство: значение;

}

### **Миксование классов**

У HTML-элемента может быть сколько угодно классов, в этом случае они перечисляются в атрибуте class через пробел, например:

<li class="product">Товар</li>

<li class="product hit">Товар, а ещё хит продаж</li>

<li class="product hit sale">Товар, хит продаж и со ски-и-идкой!</li>

Обычно миксование используют так: в один класс выносят общее оформление, а в дополнительных классах описывают его модификации.

### **Комментарии**

В CSS тоже существуют комментарии, их отличие от HTML-комментариев в том, что код или подсказки пишутся между символами /\* и \*/.

Начнём!

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

<!DOCTYPE html>

Простейшая HTML-страница состоит как минимум из трёх тегов: <html>, <head> и <body>.

Тег <html> располагается в документе сразу после доктайпа и содержит все остальные теги, включая <head> и <body>. Тегу <html> обычно добавляют важный атрибут lang (сокращение от английского «language»), в котором задаётся язык документа. Наш сайт будет русскоязычным, поэтому в lang пропишем такое значение:

<html lang="ru">

Тег <head> хранит важную служебную информацию, а в теге <body> хранится содержание страницы, которое отображается в окне браузера. Все тексты и картинки мы будем добавлять внутрь <body>. В документе теги head и body могут быть использованы только один раз.

Давайте начнём работу над прототипом главной страницы нашего сайта!

Атрибут lang можно задавать любым тегам, если нужно уточнить, на каком языке написан текст внутри тега. Но если задать его тегу <html>, то действие атрибута распространится на весь документ целиком.

Тег <head> предназначен для хранения служебной информации о странице. Он располагается первым в теге <html>, сразу перед <body>.

Внутри <head> обычно содержится заголовок, ключевые слова, описание страницы и другие служебные данные. Также внутри него подключаются внешние ресурсы, например, стили. Содержимое этого тега не отображается на странице напрямую.

Помните, в [первой части](https://htmlacademy.ru/courses/297/) стартового тренажёра мы уже пробовали менять содержимое тега <head>? Тогда мы с помощью тега <link> подключали к документу разные CSS-файлы:

<head>

<link href="адрес\_файла\_стилей.css" rel="stylesheet">

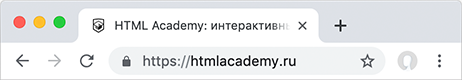
</head>

У <link> в атрибуте href задаётся адрес стилевого файла, а атрибут rel со значением stylesheet говорит браузеру, что мы подключаем именно стили, а не что-то другое.

До этого мы подключали полностью готовые стили дизайна, теперь же подключим специальные стили для прототипирования. Они «проявят» крупные блоки и немного изменят оформление текста. С этими стилями нам будет удобнее проектировать сайт, так как будет видна структура всей страницы.

Обычно стили подключаются внутри <head>, но это необязательно. Вы также можете подключить стили внутри <body>. Это не будет ошибкой, хотя делать так не рекомендуется.

Ещё один элемент, который располагается в <head> — это тег <title>. В нём задаётся заголовок страницы, который отображается во вкладках браузера:



А вот пример использования <title> в коде:

<head>

<title>Тренажёры — HTML Academy</title>

</head>

Текст в <title> должен описывать содержимое документа в целом. По заголовку должно быть понятно, о чём эта страница, даже когда она не открыта в браузере, а отображается в результатах поиска или в браузерных закладках.

Давайте дадим нашей странице подходящее название. Вы увидите, как изменится заголовок страницы в верхней части мини-браузера.

**Тег main, основное содержание**

Давайте временно переключимся со служебной информации на содержание. Сейчас мы проектируем структуру страниц на уровне крупных блоков (введение, основное содержание, заключение и так далее) и для этого нам нужны подходящие теги.

И первый такой тег — это тег <main>. Он выделяет основное содержание страницы, которое не повторяется на других страницах. И обычно на странице используется один <main>.

<main>

Привет, я основной контент! Живу только на этой странице.

</main>

Наш сайт будет состоять из нескольких страниц. На каждой из них будут повторяющиеся шапка и подвал, а также уникальный контент страницы. Вот его и заключим в тег <main>.

На главной странице, которую мы сейчас верстаем, есть приветственный текст. Он является её основным контентом и на других страницах его не будет. А значит весь этот текст нужно заключить в тег <main>. На внутренних страницах в <main> будет уже другое основное содержимое.

Спецификация не допускает использование на одной странице более одного тега <main>, если у них нет специального атрибута hidden. Этот атрибут добавляется HTML-элементу, например, в одностраничных приложениях (Single Page Application), чтобы менять содержимое страницы, делая видимым тот или иной <main> в разных состояниях приложения. Атрибут hidden указывает браузеру, что элемент не должен отображаться и использоваться в момент, когда отображается и используется содержимое другого <main>.

Кроме уникального основного содержимого страницы, у нас есть повторяющиеся на других страницах вводная часть и заключительная часть.

Вводную часть страницы, которую чаще называют «шапкой», описывает тег <header>. Аналогично ему, заключительную часть страницы, или «подвал», описывает тег <footer>.

<header>

Я шапка сайта. Могу повторяться на других страницах.

</header>

<main>

Я основной контент! Живу только на этой странице.

</main>

<footer>

Я подвал сайта, я как шапка.

</footer>

Обычно на странице появляется по одному тегу <header> и <footer>, но их может быть и больше.

Давайте разметим шапку и подвал главной страницы. В качестве содержания пока используем текстовые «заглушки».

Тег <header> — это не только шапка сайта с логотипом и меню, он может использоваться, например, и как «шапка» какой-нибудь статьи или раздела сайта. Конечно, в случае со статьёй <header> называют не «шапкой», а вводной частью, в которой могут содержаться заголовки, оглавление и так далее.

С <footer> ситуация аналогичная. В привычном понимании это подвал сайта, с копирайтами, контактной информацией и так далее. Но <footer> может использоваться и в других разделах сайта. Например, внутри статьи в «подвале» можно разместить дополнительную информацию: данные об авторе, дополнительные ссылки и так далее.

# Тег nav, основная навигация

Помните о задании инструктора Кекса? Мы разрабатываем сайт, на котором будет главная страница и записи блога. Поэтому нам нужно предусмотреть на главной странице навигационный блок, с которого можно будет перейти на определённые записи.

Для создания логического раздела с основной навигацией предназначен тег <nav> (сокращение от английского «navigation»). Обычно в <nav> включают ссылки на другие страницы или навигацию по текущей странице. Пример:

<nav>

Первый пост, второй пост, архив постов

</nav>

Со ссылками мы разберёмся в следующих частях. А пока что добавим навигационный раздел на главной странице. Этот раздел на главной будет особенным и на других страницах повторяться не будет, поэтому расположим его в основном содержании.

Не каждая группа ссылок на странице должна быть обёрнута в <nav>. Например, небольшой блок со вспомогательными ссылками в подвале сайта. Такой блок внутри тега <footer> не нужно дополнительно оборачивать в тег <nav>.

Кроме того, блок <nav> помимо ссылок может включать абзацы с текстом, заголовки, списки и другое содержание.

# Тег article, независимый раздел

Отлично, с крупными блоками главной страницы разобрались. Можем переходить к внутренней странице с записью блога. На ней будут располагаться уже привычные шапка, основное содержание и подвал.

Шапка у внутренних страниц будет повторяться: в ней будет находиться блок навигации со ссылкой на главную. На главной <nav> был уникальным и попал в <main>, на внутренних <nav> повторяется, поэтому мы поместим его в <header>.

Внутри <main> на внутренней пока располагается только пост, но позже там могут появиться другие разделы, например, облако тегов. Поэтому пост сразу нужно выделить каким-то тегом. Может для этого подойдёт уже знакомый нам тег <section>? Да, подойдёт, но есть кое-что получше!

И это тег <article>, который обозначает цельный, законченный и самостоятельный фрагмент информации. А пост в блоге именно такой.

Тег <article>, в отличие от <section>, можно вырвать из одного места и вставить в другое (на другую страницу сайта или на другой сайт), и смысл содержимого тега при этом не потеряется. Примеры: статья, пост в блоге, сообщение на форуме и так далее.

<article>

Я фотка в Инстаграме, смотрюсь отлично где угодно

</article>

Теги <section> можно использовать внутри <article>, если там нужно выделить отдельные смысловые блоки.

Точно так же можно использовать <article> внутри <section>, если в логическом разделе документа содержатся независимые контентные блоки.

# Тег aside, дополнительное содержание

Помимо <section> и <article> есть ещё один крупный логический контейнер. Это тег <aside>.

Тег <aside> включает в себя дополнительное содержание, не связанное напрямую с основным. Такие блоки ещё часто называют «сайдбарами» или боковыми панелями.

<aside>

Я скромный блок с курсами валют на сайте про рыбок

</aside>

На нашей внутренней странице тоже стоит предусмотреть <aside>. Позже мы сможем включить туда ссылки на похожие посты блога или ленту постов из Твиттера, или что-то подобное (кто знает, что придёт в голову боссу).

В стилях для прототипирования мы по-разному оформили теги, которые применяются для решения разных задач. При этом мы опирались на [систему типов](https://html.spec.whatwg.org/multipage/dom.html#kinds-of-content) спецификации HTML. Вот расшифровка:

* Пунктиром выделен особый тег <body>.
* Синей рамкой выделяются [поточные теги](https://html.spec.whatwg.org/multipage/dom.html#flow-content), которыми обычно размечают крупные структурные блоки страниц, например <main>.
* Фиолетовой рамкой выделяются [теги для создания смысловых разделов](https://html.spec.whatwg.org/multipage/dom.html#sectioning-content), например <section>.
* Оранжевой рамкой выделяются [заголовочные теги](https://html.spec.whatwg.org/multipage/dom.html#heading-content), например <h1>.
* Розовой рамкой выделяются [поточные теги](https://html.spec.whatwg.org/multipage/dom.html#flow-content), которыми обычно размечают непосредственно текстовые элементы, например <p>.

Конечно, в спецификации больше типов, но мы не стали оформлять все, а ограничились только теми тегами, которые понадобятся нам в прототипе.

# Теги h1-h6, заголовки в HTML

Мы закончили прототипировать крупные блоки страниц. Теперь давайте вернёмся к главной странице и немного поработаем над структурой текстового содержания.

Для создания основной структуры текста используют заголовки. В HTML существует целое семейство заголовочных тегов: от <h1> до <h6>. Тег <h1> обозначает самый важный заголовок (заголовок верхнего уровня), а тег <h6> обозначает подзаголовок самого нижнего уровня. Буква «h» в названии тега — это первая буква английского «heading».

На практике в текстах редко встречаются подзаголовки ниже третьего уровня. Поэтому чаще всего используются теги <h1>, <h2> и <h3>:

<h1>Спецификация HTML</h1>

<h2>Раздел 1 Введение</h2>

<h3>Раздел 1.1 Происхождение языка</h3>

Поисковые системы придают особое значение заголовкам, также правильно расставленные заголовки важны для доступности документа. Поэтому нужно учиться грамотно использовать заголовки.

Заголовок <h1> — самый важный на странице. В него нужно включать текст, который в целом описывает содержание страницы. Очень важно, чтобы заголовок первого уровня на странице был только один.

На главных страницах заголовок верхнего уровня часто добавляют в шапку сайта. В нашем прототипе мы поступим так же.

В пятой версии HTML разрешили использовать собственную, независимую от остального документа, иерархию заголовков в [тегах для создания смысловых разделов](https://html.spec.whatwg.org/multipage/dom.html#sectioning-content). Теперь на странице можно использовать несколько <section> или <article> со своими <h1>, <h2> и <h3>.

На практике выяснилось, что этот механизм скорее мешает, чем помогает, а браузеры и средства доступности не спешат его реализовывать. Так что все начали возвращаться к старой доброй сквозной иерархии заголовков во всём документе.

# Тег p, параграф

Основную структуру текста создают с помощью заголовков, а более мелкую выстраивают с помощью параграфов (или абзацев). Для разметки параграфов предназначен тег <p> (от английского «paragraph»).

По умолчанию абзацы начинаются с новой строки и отделяются от остального контента отступами сверху и снизу. Так что, если нужно отделить один блок текста от другого, верным решением будет заключить их в теги <p>.

<p>Абзац про то, как я решил стать верстальщиком</p>

<p>Абзац про моего инструктора</p>

<p>Абзац про мой дневничок</p>

Давайте теперь разметим абзацами приветственный текст на нашей главной странице.

Абзац — отрезок письменной речи, состоящий из одного или нескольких предложений, зачастую объединённых общей мыслью. Это привычное нам определение. Параграф в HTML совсем другой.

Параграфы в HTML — это всего лишь неразрывная последовательность [фразовых](https://html.spec.whatwg.org/multipage/dom.html#phrasing-content) элементов, то есть чисто структурная, а не смысловая сущность. Параграфы существуют даже без тега <p>.

А тег <p> позволяет лишь явно выделять параграфы, группируя элементы с *фразовым* типом содержимого. И это могут быть не только блоки текста, а, например, изображения, ссылки или поля ввода.

Не все теги могут быть включены в <p>. Например, внутри абзаца не могут располагаться крупные структурные теги, заголовки, формы, списки, таблицы. Когда браузер встречает неподходящий тег внутри <p>, он «выбрасывает» этот тег из <p>.

# Тег meta, кодировка страницы

Отлично! С разметкой контента мы закончили. Снова возвращаемся к тегу <head>.

Важный тег, который включается в <head> — тег <meta>. Он одиночный, то есть не требует парного закрывающего тега в конце.

С помощью <meta> можно сообщать браузеру, поисковому роботу или другому устройству различную служебную информацию (или *метаинформацию*) о вашем сайте: кодировку текста, описание контента и так далее. Для этого используются теги <meta> с разными атрибутами и их значениями. Вот некоторые из атрибутов: charset, content, http-equiv, name и scheme.

С помощью атрибута charset указывается кодировка текста HTML-страницы:

<meta charset="название кодировки">

Лучше всегда указывать кодировку явно. Если этого не делать, браузер может неправильно угадать её, и вместо текста будут отображаться «иероглифы».

Самая распространённая современная кодировка — utf-8. Используйте её во всех своих проектах. Раньше часто использовали кодировку windows-1251, стандартную кодировку для кириллицы в Windows. Но сейчас это считается плохой практикой.

<meta>, <link>, <title> и другие теги, включаемые в <head>, имеют особый тип содержимого — *метасодержимое*. Они не отображаются на странице напрямую, а служат для хранения информации о документе и для взаимосвязи документа с другими документами и системами.

# Тег meta, кодировка страницы

Отлично! С разметкой контента мы закончили. Снова возвращаемся к тегу <head>.

Важный тег, который включается в <head> — тег <meta>. Он одиночный, то есть не требует парного закрывающего тега в конце.

С помощью <meta> можно сообщать браузеру, поисковому роботу или другому устройству различную служебную информацию (или *метаинформацию*) о вашем сайте: кодировку текста, описание контента и так далее. Для этого используются теги <meta> с разными атрибутами и их значениями. Вот некоторые из атрибутов: charset, content, http-equiv, name и scheme.

С помощью атрибута charset указывается кодировка текста HTML-страницы:

<meta charset="название кодировки">

Лучше всегда указывать кодировку явно. Если этого не делать, браузер может неправильно угадать её, и вместо текста будут отображаться «иероглифы».

Самая распространённая современная кодировка — utf-8. Используйте её во всех своих проектах. Раньше часто использовали кодировку windows-1251, стандартную кодировку для кириллицы в Windows. Но сейчас это считается плохой практикой.

<meta>, <link>, <title> и другие теги, включаемые в <head>, имеют особый тип содержимого — *метасодержимое*. Они не отображаются на странице напрямую, а служат для хранения информации о документе и для взаимосвязи документа с другими документами и системами.

# Тег meta, ключевые слова

С помощью метатегов можно добавить на страницу информацию полезную для поисковых систем: перечень ключевых слов и краткое описание страницы.

Перечень ключевых слов задаётся тегом <meta>, у которого атрибут name имеет значение keywords. Ключевые слова (самые важные слова из содержания страницы) перечисляются в атрибуте content через запятую:

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

Раньше этот тег был очень важен для поисковиков. Каково положение дел сейчас? Мы бы с удовольствием вам поведали, но это большой секрет Яндекса и Гугла.

Краткое описание страницы задаётся похожим образом, только значение атрибута name меняется на description:

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

Краткое описание (или аннотация) страницы часто используется поисковиками при отображении результатов поиска.

<!DOCTYPE html>

<html lang="ru">

<head>

<meta charset="utf-8">

<meta name="keywords" content="вёрстка, HTML, CSS, блог">

<meta name="description" content="Блог о процессе обучения веб-технологиям">

<title>Сайт начинающего верстальщика</title>

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

</head>

<body>

<header>

<h1>Сайт начинающего верстальщика</h1>

</header>

<main>

<nav>

Навигация

</nav>

<section>

<p>Всем привет! Добро пожаловать на мой первый сайт. Ещё недавно я понятия не имел, кто такой верстальщик, а теперь я нашёл тренажёры по HTML и CSS и поставил перед собой цель — стать им. У меня даже появился инструктор — Кекс, который не позволит мне расслабиться и будет следить за моими успехами.</p>

<p>Моё первое задание — вести дневник и честно писать обо всех своих свершениях.</p>

</section>

<section>

Раздел про навыки

</section>

</main>

<footer>

Подвал сайта

</footer>

</body>

</html>

# Конспект «Структура HTML-документа»

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

<!DOCTYPE html>

Простейшая HTML-страница состоит как минимум из трёх тегов: <html>, <head> и <body>. Тег <head> обычно содержит заголовок, ключевые слова, описание страницы и другие служебные данные. Также внутри него подключаются внешние ресурсы, например, стили. Содержимое этого тега не отображается на странице напрямую. А в теге <body> хранится содержание страницы, которое отображается в окне браузера.

Для подключения стилей к странице существует тег <link>. Для этого у него есть атрибут href в котором задаётся адрес стилевого файла, а значение stylesheet атрибута rel говорит браузеру, что мы подключаем именно стили, а не что-то другое.

<head>

<link href="адрес\_файла\_стилей.css" rel="stylesheet">

</head>

Ещё один элемент, который располагается в <head> — это тег <title>. В нём задаётся заголовок страницы, который отображается во вкладках браузера. По заголовку должно быть понятно, о чём эта страница, даже когда она не открыта в браузере, а отображается в результатах поиска или в браузерных закладках.

<head>

<title>Тренажёры — HTML Academy</title>

</head>

Ещё один важный тег, располагающийся внутри <head> это тег <meta>. Он одиночный, то есть не требует парный закрывающий тег в конце. С помощью <meta> можно сообщать браузеру, поисковому роботу или другому устройству различную служебную информацию (или метаинформацию) о вашем сайте: кодировку текста, описание контента и так далее. Для этого используются теги <meta> с разными атрибутами и их значениями.

Кодировка текста HTML-страницы указывается с помощью атрибута charset:

<meta charset="название кодировки">

Самая распространённая современная кодировка — utf-8.

Перечень ключевых слов задаётся тегом <meta>, у которого атрибут name имеет значение keywords. Ключевые слова (самые важные слова из содержания страницы) перечисляются в атрибуте content через запятую:

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

Краткое описание (или аннотация) страницы задаётся похожим образом, только значение атрибута name меняется на description:

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

Внутри <body> находятся те теги, которые отображаются на странице. Например, тег <main> выделяет основное содержание страницы, которое не повторяется на других страницах. И обычно на странице используется один <main>.

Тег <header> содержит вводную часть страницы, которую чаще называют «шапкой», а тег <footer> описывает заключительную часть страницы, или «подвал». Существует тег <section>, который обозначает крупный смысловой (или «логический») раздел.

Тег <article>, обозначает цельный, законченный и самостоятельный фрагмент информации.

Для создания логического раздела с основной навигацией предназначен тег <nav> (сокращение от английского «navigation»). Обычно в <nav> включают ссылки на другие страницы или навигацию по текущей странице.

Тег <aside> включает в себя дополнительное содержание, не связанное напрямую с основным. Такие блоки ещё часто называют «сайдбарами» или боковыми панелями.

Для создания основной структуры текста используют заголовки. В HTML существует целое семейство заголовочных тегов: от <h1> до <h6>. Тег <h1> обозначает самый важный заголовок (заголовок верхнего уровня), а тег <h6> обозначает подзаголовок самого нижнего уровня.

<h1>Спецификация HTML</h1>

<h2>Раздел 1 Введение</h2>

<h3>Раздел 1.1 Происхождение языка</h3>

Основную структуру текста создают с помощью заголовков, а более мелкую выстраивают с помощью параграфов (или абзацев). Для разметки параграфов предназначен тег <p>. По умолчанию абзацы начинаются с новой строки и отделяются от остального контента отступами сверху и снизу.

Мы дополнили запись и процитировали целое стихотворение. И это не какая-то короткая цитата внутри абзаца, поэтому тег <q> не поможет.

Для выделения длинных цитат, которые могут состоять из нескольких абзацев, предназначен тег <blockquote>. Он выделяет цитату не как фрагмент текста в предложении, а как отдельный блок текста с отступами.

Так же, как и у <q>, у <blockquote> может быть атрибут cite, содержащий ссылку на источник цитаты. И также в паре с <blockquote> можно использовать тег <cite>, который обычно размещается внутри цитаты. Пример:

<blockquote>

<p>Ум ценится дорого, когда дешевеет сила.</p>

<cite>Джейсон Стэтхэм</cite>

</blockquote>

В браузере тегу <blockquote> обычно добавляются горизонтальные отступы.

Мы добавили немного стилей (отступы, рамку и цвет текста), чтобы содержимое <blockquote> смотрелось красивее. Вы можете изучить стили во вкладке style.css.

# Тег br, перенос строки

Стихотворение в цитате всё ещё не похоже на стихотворение, так как не хватает переносов строк внутри строфы. Мы уже знаем много тегов, может какие-то из них подойдут для разбиения текста на строки?

Первый кандидат — тег <p>. Идеально подходит для выделения строфы, а нам нужно добавить переносы внутрь строф. Отметаем вариант. Второй кандидат — тег <pre>. Сохраняет всё форматирование текста. Вроде подходит, но нам нужно сохранить только переносы строк. В общем, ничего не подходит.

Нужен специальный тег, который просто добавляет перенос строки внутри абзаца. Для этого в HTML предусмотрен одиночный тег <br> (сокращение от «line break»).

Этот тег отлично подходит, если переносы нужны в тексте для повышения его читабельности, например, в почтовых адресах, стихах, текстах песен, режиме работы.

Иногда этот тег применяют неправильно, для разбиения текста на «как бы абзацы». Никогда так не делайте. Для разметки абзацев нужно использовать тег <p>.

# Теги sub и sup, нижний и верхний индексы

Наконец-то новая запись. Вам доверили поработать с методическим материалом, в котором описываются формулы. Всё почти завершено, осталось только расставить верхние и нижние индексы. Для них тоже есть специальные теги.

Тег <sup> (сокращение от «superscript») отображает текст в виде верхнего индекса, а тег <sub> (сокращение от «subscript») в виде нижнего индекса.

Эти теги чаще используются для выделения отдельных символов, а не слов. Их используют для указания единиц измерения или написания простых формул. Например:

20м<sup>2</sup>

H<sub>2</sub>O

X<sup>3</sup>+X<sup>2</sup>=1

В браузере это будет выглядеть так:

20 м2  
H2O  
X3+X2=1

Для создания более сложных формул, например, ex2 или уxn+1, теги <sup> и <sub> можно использовать внутри друг друга.

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

# Тег time, дата и время

В новой записи вы решили уточнить точное время дня (а точнее ночи), когда именно была сделана запись. Строку со временем можно оставить просто текстом, но есть вариант получше. В HTML есть специальный тег для разметки даты и времени. Это тег <time>.

С помощью <time> можно описывать даты одновременно и для человека, и для машины. Дата для человека описывается внутри тега, а дата для машины — внутри атрибута datetime в формате ISO 8601:

<time datetime="2016-11-18T09:54">09:54 утра</time>

<time datetime="2015-11-18">18 ноября 2015</time>

<time datetime="2018-09-23">в прошлую субботу</time>

<time datetime="2017-04-20">вчера</time>

Браузер отображает только содержимое тега, а содержимое datetime не отображается. Человек увидит только понятное ему обозначение времени, а машина прочитает атрибут и получит дату в нужном ей формате. И все останутся довольны.

Мы добавили немного стилей, чтобы теги <time> смотрелись красивее. Вы можете изучить стили во вкладке style.css.

# Теги em и i, акцентирование внимания

Время идёт, ваш дневник пополняется. Со временем вы замечаете, что хотите выделить некоторые фрагменты текста ваших записей. И не просто выделить, а сделать на них *особый акцент*.

Следующие два тега предназначены для акцентирования внимания на слово или фразу. Это теги <em> (сокращение от «emphasis») и <i> (сокращение от «italic»).

Визуально оба тега одинаковы, они выделяют текст курсивом. Но значение у этих тегов различное.

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

Например, если автор текста любит лето и недолюбливает зимнее время, то его речь можно разметить следующим образом:

Я <em>просто обожаю</em> холодные зимние дни!

Тег <i> применяется для обозначения текста, который отличается от окружающего текста, но не является более важным. Например, в <i> можно заключать *названия*, *термины*, *иностранные слова*. Также в этот тег можно обернуть *мысли* героя. В речи такой текст обычно выделяется интонационно. Приведём пример:

Он взглянул в окно и подумал — <i>такого просто не может быть</i>!

# Теги strong и b, важность и выделение

На этом возможности выделения слов и фраз в тексте не заканчиваются. Вы научились выделять фразы курсивом, а теперь давайте разберёмся с выделением текста жирным.

Теги <strong> и <b> (сокращение от «bold») предназначены для **выделения** слова или фразы. Отображаются оба тега одинаково, они выделяют текст жирным.

Тег <strong> указывает на **важность** отмеченного текста. Он может использоваться для выделения предупреждений или части документа, которую пользователь должен увидеть раньше остального. При этом обозначение части текста тегом <strong> не должно изменять смысла предложения.

<strong>Внимание!</strong> Это место опасно. <strong>Вы можете упасть в пропасть</strong>, если подойдёте близко к краю.

Тег <b> предназначен для выделения текста с целью привлечения к нему внимания, но без придания ему особой важности. Использовать его нужно только в случае, когда остальные теги выделения не подходят. Типичный пример — выделение вводного предложения статьи.

Вы входите в небольшую комнату. Ваш <b>меч</b> загорается ярче. <b>Крыса</b> стремительно пробегает вдоль стены.

Лучше всего отличия этих тегов будут заметны людям, которые используют средства доступности, в частности, слепым и слабовидящим. Скринридер при чтении сайта будет выделять слова с тегом <strong> интонационно, в отличие от простого выделения с помощью <b>.

То же самое касается тегов <em> и <i>. Тег <em> «читалка» будет выделять интонацией.

# Теги del и ins, выделение изменений

Списки дел на то и нужны, чтобы вычёркивать выполненные дела и иногда добавлять новые. У вас также назрела необходимость обновить список дел, но так, чтобы было видно, какие дела завершены, а какие добавлены. Как сделать разметку такой истории изменений?

Как раз для описания изменений предназначены теги <del> (сокращение от «delete») и <ins> (сокращение от «insert»).

Тег <del> выделяет текст, который был удалён в новой версии документа. В браузере этот текст перечёркивается.

Тег <ins> выделяет текст, который был добавлен в новой версии документа. В браузере этот текст подчёркивается.

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 дел. Когда дело выполнено, его помечают тегом <del>, а если появилось новое дело, то его добавляют в список и помечают тегом <ins>.

<ul>

<li>Почистить посудомоечную машину</li>

<li><del datetime="2009-10-11T01:25-07:00">Погулять</del></li>

<li><del datetime="2009-10-10T23:38-07:00">Поспать</del></li>

<li><ins>Купить принтер</ins></li>

</ul>

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.

# Теги div и span, контейнеры для стилизации

Вы уже узнали так много новых тегов и у каждого из них есть свой смысл. Но иногда хочется просто красиво, без придания какого-то смысла, выделить короткий фрагмент текста или несколько абзацев. Например, подсветить самые опасные на ваш взгляд болезни верстальщика красноватым фоном.

Но раз такое выделение «бессмысленно», то и делать его «смысловыми» тегами нельзя? Да, это так.

Но есть два специальных тега у которых «смысла» нет. Это теги <div> (сокращение от «division») и <span>. Это «чистые» элементы, которые отлично подходят для визуальной группировки других элементов. Использовать эти теги рекомендуется, если более подходящих семантических тегов не нашлось.

Теги <div> и <span> не имеют никакого оформления по умолчанию и их почти всегда используют вместе с атрибутом class, чтобы легко добавлять им собственные стили.

Тег <div> обычно используется для группировки крупных элементов, например, нескольких абзацев, или в качестве контейнера для создания сеток страниц. А <span> используется для выделения мелких текстовых элементов: частей слов, отдельных слов или фраз, состоящих из нескольких слов:

<article>

<div class="highlight">

<p>Мы два красивых выделенных абзаца.</p>

<p>С жёлтеньким фоном!</p>

</div>

<p>Текст, в котором <span>выделена фраза</span>.</p>

</article>

В этом задании мы используем <div> в качестве обёртки для пунктов списка определений. В таком случае в теге <dl> не должно находиться не обёрнутых в <div> пунктов списка. Мы **либо** оборачиваем все элементы, **либо** оставляем все теги <dt> и <dd> без обёрток. Смешение обёрнутых и не обёрнутых в <div> определений внутри одного списка считается ошибкой.

После выполнения этого шага, можно скачать промежуточное состояние нашего Сайта начинающего верстальщика по этой [ссылке](https://htmlacademy.ru/assets/courses/301/project-state-2.zip).

Мы подготовили стили для выделения текста и блока. Обратите внимание на вкладку style.css.

# Конспект «Разметка текста»

## Списки

### **Неупорядоченный список**

Тег <ul> (сокращение от «unordered list»). Используется, когда порядок элементов не важен. Например, для разметки перечня ссылок в меню, преимуществ товара, ингредиентов в составе продукта.

Непосредственно в теге <ul> могут находиться только теги <li> (сокращение от «list item»), которые обозначают элементы или пункты списка:

<ul>

<li>Я пункт списка, могу быть на любом месте</li>

<li>И я пункт списка, и мне тоже не важен порядок</li>

</ul>

По умолчанию элементы <ul> отмечаются маркерами такого же цвета, как цвет текста.

* Я пункт списка, могу быть на любом месте
* И я пункт списка, и мне тоже не важен порядок

### **Упорядоченный список**

Тег <ol> (сокращение от «ordered list»). В этом списке действительно важно, в каком порядке идут элементы. Упорядоченные списки подходят для разметки алгоритмов, инструкций, рецептов, результатов соревнований и так далее.

Пункты упорядоченного списка тоже размечаются с помощью тега <li>. Пример кода:

<ol>

<li>Я первый и только первый пункт</li>

<li>Я не я, если я не второй пункт</li>

<li>Третий после стольких лет? Всегда!</li>

</ol>

По умолчанию перед элементами <ol> ставится их порядковый номер.

1. Я первый и только первый пункт
2. Я не я, если я не второй пункт
3. Третий после стольких лет? Всегда!

У <ol> может быть несколько атрибутов: start, reversed и type.

Атрибут start меняет стартовое число нумерации пунктов. Может быть отрицательным.

Атрибут reversed меняет направление нумерации на противоположный. Этот атрибут не требует значения.

С помощью атрибута type можно задавать различные типы маркеров: строчные и заглавные латинские буквы или римские цифры.

### **Список описаний**

Тег <dl> (сокращение от «description list»). Список описаний используется для разметки вопросов-ответов, наименований и определений, категорий и тем. Он создаётся с помощью трёх тегов:

* <dl> обозначает сам список описаний;
* <dt> (сокращение от «description term») обозначает термин;
* <dd> (сокращение от «description definition») обозначает описание или определение.

Теги <dt> и <dd> пишутся внутри <dl>. Каждый список <dl> может содержать один или несколько терминов и одно или несколько описаний для каждого термина. Пример кода:

<dl>

<dt>HTML</dt>

<dd>Язык гипертекстовой разметки</dd>

<dt>CSS</dt>

<dd>Каскадные таблицы стилей</dd>

<dd>Язык для оформления HTML-документов</dd>

</dl>

По умолчанию браузер добавляет небольшой отступ слева от определений.

HTML

Язык гипертекстовой разметки

CSS

Каскадные таблицы стилей

Язык для оформления HTML-документов

## Преформатированный текст и код

Тег <pre> (сокращение от «preformatted text»). Используется для отображения примеров кода, также применяется для отображения картинок ASCII Art. Браузер сохраняет и отображает все пробелы и переносы, которые есть внутри тега <pre>.

<pre>Пример

преформатированного

текста с сохранёнными пробелами

и переносами строк</pre>

Пример

преформатированного

текста с сохранёнными пробелами

и переносами строк

Тег <code>. Используется для обозначения фрагментов кода.

С его помощью размечается любой фрагмент текста, который распознается компьютером: код программы, разметки, название файла и так далее. Обычно браузеры отображают текст в теге <code> моноширинным шрифтом.

Тег <code>ul</code> — это неупорядоченный список.

Тег ul — это неупорядоченный список.

Тег <code> можно вкладывать внутрь тега <pre>.

## Цитаты

### **Небольшие цитаты**

Тег <q> (сокращение от «quote»). Предназначен для выделения цитат внутри предложения. Текст внутри тега браузер автоматически обрамляет кавычками, поэтому добавлять кавычки вручную не нужно.

### **Источник цитат**

Тег <cite>. В нём можно указывать помимо адреса источника цитаты ещё и название произведения, откуда цитируется текст, а также имя автора или организации, чей текст цитируется. Содержимое <cite> в браузере выделяется курсивом.

<p>По словам <cite>Чарльза Буковски</cite> — <q>Интеллектуал о простой вещи говорит сложно — художник сложную вещь описывает простыми словами.</q></p>

По словам Чарльза Буковски — Интеллектуал о простой вещи говорит сложно — художник сложную вещь описывает простыми словами.

Тег <cite> может быть самостоятельным и не привязываться к цитате:

<p>Какой доктор ваш любимый (в сериале <cite>Доктор Кто</cite>)?</p>

### **Длинные цитаты**

Тег <blockquote>. Предназначен для выделения длинных цитат, которые могут состоять из нескольких абзацев. Тег выделяет цитату не как фрагмент текста в предложении, а как отдельный блок текста с отступами.

<blockquote>

<p>Ум ценится дорого, когда дешевеет сила.</p>

<cite>Джейсон Стэтхэм</cite>

</blockquote>

В браузере контенту тега <blockquote> обычно добавляется дополнительный отступ слева и справа.

Обычный текст.

Ум ценится дорого, когда дешевеет сила.

Джейсон Стэтхэм

## Разметка фрагментов текста

### **Символы-мнемоники**

Это особые строки, которые начинаются с амперсанда (&) и заканчиваются точкой с запятой (;). Например, знак *меньше* на страницу можно вставить мнемоникой &lt; (less than), а знак *больше* мнемоникой &gt; (greater than):

Некоторые символы в HTML зарезервированы, то есть браузер считает их HTML-кодом. Например, любой текст после знака *меньше* (<) браузер будет пытаться интерпретировать как тег и на странице не отобразит. Чтобы использовать специальные символы в тексте страницы как обычные символы их нужно заменить на символы-мнемоники.

&lt;**ul**&gt;

&lt;/**ul**&gt;

<ul>  
</ul>

### **Перенос строк**

Тег <br> (сокращение от «line break»). Применяется, чтобы вставить в текст перенос строки, не создавая при этом абзац. Например, при разметке стихов или текстов песен.

### **Верхний и нижний индексы**

Теги <sup> и <sub>. Названия образованы от слов «superscript» и «subscript».

Тег <sup> отображает текст в виде верхнего индекса, а <sub> отображает текст в виде нижнего индекса.

Их используют для указания единиц измерения или для написания простых формул:

20м<sup>2</sup>

H<sub>2</sub>O

X<sup>3</sup>+X<sup>2</sup>=1

20м2  
H2O  
X3+X2=1

Для создания более сложных формул, эти теги можно использовать внутри друг друга.

### **Дата и время**

Тег <time>. С помощью него можно описывать даты одновременно и для человека, и для машины. Для указания даты в машиночитаемом формате ISO 8601 существует атрибут datetime и выглядит так:

<time datetime="2016-11-18T09:54">09:54 утра</time>

<time datetime="2015-11-18">18 ноября 2015</time>

<time datetime="2018-09-23">в прошлую субботу</time>

<time datetime="2017-04-20">вчера</time>

Браузер отображает только содержимое тега, а содержимое datetime не отображается.

### **Акцентирование внимания**

Теги <em> и <i>. Названия образованы от слов «emphasis» и «italic». Предназначены для акцентирования внимания на слово или фразу. Визуально оба тега одинаковы, они выделяют текст курсивом.

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

Я <em>просто обожаю</em> холодные зимние дни!

Тег <i> применяется для обозначения текста, который отличается от окружающего текста, но не является более важным. Например, в <i> можно заключать *названия*, *термины*, *иностранные слова*. Также в этот тег можно обернуть *мысли* героя. В речи такой текст обычно выделяется интонационно:

Он взглянул в окно и подумал — <i>такого просто не может быть</i>!

### **Выделение и придание важности**

Теги <strong> и <b>. Название <b> образовано от слова «bold». Отображаются оба тега одинаково, они выделяют текст жирным.

Тег <strong> указывает на **важность** отмеченного текста. Он может использоваться для выделения предупреждений или части документа, которую пользователь должен увидеть раньше остального. При этом обозначение части текста тегом <strong> не должно изменять смысла предложения.

<strong>Внимание!</strong> Это место опасно. <strong>Вы можете упасть в пропасть</strong>, если подойдёте близко к краю.

Тег <b> предназначен для выделения текста с целью привлечения к нему внимания, но без придания ему особой важности. Использовать его нужно только в случае, когда остальные теги выделения не подходят. Типичный пример — выделение вводного предложения статьи.

Вы входите в небольшую комнату. Ваш <b>меч</b> загорается ярче. <b>Крыса</b> стремительно пробегает вдоль стены.

### **Описание изменений**

Теги <del> и <ins>. Названия тегов образованы от слов «delete» и «insert». Предназначены для описания изменений в документе.

Тег <del> выделяет текст, который был удалён в новой версии документа. В браузере этот текст перечёркивается.

Тег <ins> выделяет текст, который был добавлен в новой версии документа. В браузере этот текст подчёркивается.

<ul>

<li>Почистить посудомоечную машину</li>

<li><del datetime="2009-10-11T01:25-07:00">Погулять</del></li>

<li><del datetime="2009-10-10T23:38-07:00">Поспать</del></li>

<li><ins>Купить принтер</ins></li>

</ul>

* Почистить посудомоечную машину
* Купить принтер

## Разделение контента

Теги <div> и <span>. Это «чистые» элементы, и обычно они отлично подходят в качестве обёртки для стилизации или группировки других элементов. Использовать эти теги рекомендуется в тех случаях, если более подходящих семантических тегов не нашлось.

Тег <div> используется для группировки структурных элементов или в качестве вспомогательных контейнеров для создания нужной раскладки.

Тег <span> используется для группировки текстовых элементов, выделения отдельных слов или фраз внутри абзацев, пунктов списка и так далее.

<article>

<div class="highlight">

<p>…</p>

<p>…</p>

</div>

<p>Текст, в котором <span>выделена фраза</span></p>

</article>

# Что такое ссылка, тег a

В перерывах между бизнес-встречами Кекс решил почитать «Блог начинающего верстальщика». Открыл главную страницу и не смог перейти с неё на другие страницы блога. Конфуз!

А всё потому, что на сайте совсем нет ссылок! И пока мы это не исправим, босс не будет читать блог: он слишком занят, чтобы открывать каждую страничку по отдельности.

Что же представляет из себя ссылка?

Обычно ссылка выглядит как подчёркнутый участок текста, щёлкая на который, вы переходите на другую страницу, открываете изображение, начинаете скачивать файл или перемещаетесь к какому-то месту на текущей странице. Если представить, что интернет это огромная сеть из множества узлов, то ссылки будут нитками, соединяющими все узлы этой сети.

Ссылки создаются с помощью очень короткого тега <a> (сокращение от «anchor»). Например, вот так:

<a href="https://htmlacademy.ru">HTML Academy</a>

Адрес ссылки задаётся в формате [URL](https://ru.wikipedia.org/wiki/URL) с помощью атрибута href (сокращение от «hyper reference»).

Пробуем создать ссылку в нашем блоге и перейти по ней

# Относительные адреса

Над блогом мы в основном работаем на своём компьютере, то есть *локально*. Ссылка на другую страницу блога должна открывать файл с нашего компьютера, а не откуда-то из интернета. Для таких ссылок нужно использовать особый адрес, например:

<a href="day-2.html">Вторая запись блога</a>

Такой адрес называется *относительным*. В нём, в отличие от «обычных» адресов, нет адреса сайта! Чтобы перейти по относительному адресу, браузер должен его «расшифровать». Для этого он обычно использует положение текущей страницы. Например, в папке c:/blog есть два файла:

c:/blog/

|-**index.html** // в браузере открыта эта страница

|-inner.html

В браузере открыта страница c:/blog/index.html, и в ней есть ссылка с относительным адресом inner.html. Чтобы перейти по этой ссылке, браузер смотрит на расположение открытой страницы и заменяет в нём последнюю часть:

c:/blog/ + inner.html // заменяем последнюю часть

c:/blog/inner.html // открываем этот файл

Относительные адреса работают не только для файлов на компьютере, но и для страниц в сети. Если выложить два файла из примера в интернет (не меняя их взаимное расположение), то ссылка всё равно будет работать. Относительный адрес inner.html на странице https://site.ru/blog/index.html расшифруется так:

https://site.ru/blog/ + inner.html // заменяем последнюю часть

https://site.ru/blog/inner.html // открываем этот адрес

# Абсолютные адреса

Относительные адреса отлично подходят, если вы делаете сайт на своём компьютере или создаёте навигацию по страницам своего же сайта. Но если нужно сделать ссылку на другой сайт в интернете, то необходимо использовать уже «обычный» адрес.

Этот «обычный» или полный адрес называется *абсолютным*. Выглядит он, например, так:

https://site.ru/blog/index.html

Абсолютные адреса содержат минимум три части: протокол, имя сервера и путь.

https: — протокол

//site.ru — имя сервера

/blog/index.html — путь

Если в адресе нет имени сервера или протокола, то это относительный адрес:

https://site.ru/blog/index.html - абсолютный адрес

//site.ru/blog/index.html - относительный адрес

/blog/index.html - относительный адрес

index.html - относительный адрес

В тексте главной страницы не хватает одной полезной ссылки — ссылки на тренажёры по вёрстке. Как раз для неё и пригодится абсолютный адрес.

# Ссылка на файл

В [вызове](https://htmlacademy.ru/courses/301/challenge/2) после [предыдущей части](https://htmlacademy.ru/courses/301) мы размечали рецепт печенья для Кекса. Он оказался настолько хорошим, что мы решили сделать его доступным для всех. Мы сохранили рецепт в формате PDF, чтобы любой желающий мог его скачать и распечатать. Осталось добавить ссылку на этот самый файл.

Когда мы переходим по ссылкам, то попадаем на другие страницы или места на странице. Но при переходе по ссылке на файл браузер предложит его скачать.

Однако, если браузер умеет обрабатывать файлы этого типа, то содержимое файла откроется прямо в браузере. Чаще всего так происходит с изображениями. Но в последнее время браузеры научились открывать PDF-файлы и многие другие.

Для того чтобы предотвратить открытие файлов прямо в браузере, у тега <a> существует атрибут download, который поможет именно скачать файл.

<a href="file.pdf" download>Браузер скачает меня, а не будет читать</a>

При скачивании или загрузке файлов со сторонних сайтов для безопасности можно использовать атрибут rel="noopener". Этот атрибут позволяет игнорировать скрипты сторонней страницы, которые могут влиять на загрузку файла. Особенно актуален этот атрибут в случае, если загрузка происходит в новой вкладке или новом окне.

# Ссылка-якорь

Мы нашли отличную статью, для размещения в блоге. Но статья длинная, и чтобы работать с ней было удобно, хорошо бы добавить к ней навигацию. Эта навигация не должна уводить читателя на другие страницы, а должна работать в пределах страницы со статьёй.

Для создания такой навигации используются ссылки-якоря. Ссылка-якорь — это обычная ссылка, в адресе которой используется символ #, после которого следует идентификатор элемента. Идентификатор создаётся с помощью атрибута id у того тега, к которому надо перейти при щелчке по ссылке. Причём сам тег может быть любым: <h1>, <section>, <p> и так далее.

Вот так выглядит адрес, состоящий из одного якоря:

<a href="#part1">Глава 1</a>

При щелчке по такой ссылке браузер найдёт на странице элемент с соответствующим атрибутом id и прокрутит окно страницы к нему.

<article id="part1">Содержание первой главы</article>

При этом перезагрузки страницы не произойдёт.

Интересно, что якорь также можно использовать и в абсолютных адресах, тогда после перехода на нужную страницу по аналогии произойдёт прокрутка к заданной части этой страницы.

Якоря отлично подойдут для создания оглавления для статьи в этом посте.

# Тег img, изображение

Нетрудно заметить, что на нашем сайте начинающего верстальщика совсем нет картинок. А ведь хорошо подобранные картинки правильного размера, которые отображаются без погрешностей — залог успешного бизнеса. Поэтому Кекс дал нам новое задание — разобраться во всех форматах и назначениях изображений.

Для этого он прислал нам макет нашего блога с картинками, мы их уже вырезали и сложили в нужную папку, осталось только подключить.

Итак, разберёмся с подключением изображений. Для этого существует тег <img>. Он одиночный, то есть, как мы уже знаем, не требует закрывающего тега. Сам по себе тег <img> бесполезен без своего атрибута src, который указывает путь к картинке.

<img src="logo.png">

Давайте подключим аватарку к главной странице.

# Форматы изображений, формат SVG

Кекс недоволен, сайт получился слишком тяжёлый. Что же произошло? Дело в том, что мы случайно подключили не тот формат изображений. Но не будем расстраиваться, давайте разберёмся, какие форматы существуют и исправим формат на подходящий.

На самом деле форматов существует достаточно много, но основных несколько: JPEG, PNG, SVG и GIF. С ними мы и познакомимся.

На этом шаге рассмотрим подробнее формат SVG. SVG (Scalable Vector Graphics) переводится как масштабируемая векторная графика. Качество таких изображений не меняется при изменении размеров, да и вес у них небольшой. Формат SVG отлично подходит для малоцветных схем, логотипов и иконок.

А ещё SVG чем-то похож на HTML: он описывается в человекочитаемом текстовом формате. То есть вы можете «кодить» SVG-изображения своими руками. Познакомиться с SVG вы можете в нашем [тренажёре](https://htmlacademy.ru/courses/svg).

То есть, формат SVG подходит если:

* необходимо масштабировать изображение без потерь;
* изменять цвет элементов изображения;
* нужно анимировать части изображения.

На следующих шагах мы разберём оставшиеся форматы, а сейчас давайте добавим аватарку в правильном формате SVG в наш блог.

**Формат JPEG**

Давайте продолжим разбираться с форматами изображений. Сейчас нам нужно вставить картинки в блог в правильном формате. Первая картинка — это фотография, и для неё прекрасно подходит формат JPEG.

Этот формат был разработан для сжатия и хранения полноцветных фотографий. Он поддерживает более 16 миллионов цветов. Но нужно помнить, что JPEG сжимает изображения с потерей качества. Поэтому если мы хотим уменьшить вес изображения, то придётся ухудшать его внешний вид. Главная задача при работе с JPEG — подобрать такой уровень сжатия, чтобы и вес, и качество картинки были приемлемыми.

Таким образом, формат JPEG лучше подходит для:

* полноцветных изображений, фотографий;
* изображений с плавным переходом яркости и контраста;
* рисунков с большим количеством разноцветных деталей.

Давайте добавим JPEG-картинку в новую запись в блоге.

**Формат PNG**

В отличие от JPEG, PNG является форматом сжатия без потерь и позволяет сохранять изображения, в которых требуется особенная чёткость. Например, скриншоты сайтов, чертежи и печатный текст. Главная особенность формата PNG — поддержка прозрачности, то есть каждому пикселю в отдельности можно задать свою степень прозрачности.

Итак, формат PNG подходит для:

* изображений с прозрачностью и полупрозрачностью;
* полноцветных изображений, когда необходима повышенная точность;
* изображений с резкими переходами цветов.

PNG — это относительно новый формат, который был введён как альтернатива для формата GIF, который мы рассмотрим на следующем шаге. А сейчас добавим PNG-картинку в блог.

Обратите внимание, что часть картинок мы подключаем из папки img, а часть — из папки files, например:

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

<img src="files/portrait.png">

В папке img мы решили хранить «интерфейсные» картинки. Обычно это изображения, которые появляются ещё при вёрстке сайта. Они могут повторяться во многих местах *интерфейса* сайта.

В папке files мы решили хранить «контентные» картинки. Это изображения, которые обычно появляются при наполнении сайта. Контент-менеджер может загружать их на сайт с помощью системы управления и потом вставлять в *содержание* отдельных страниц.

Разделять эти ресурсы полезно. Например, мы сможем запретить менеджеру работать с папкой img, и он не сможет по ошибке удалить какую-то важную интерфейсную картинку.

**Формат GIF**

И последний формат изображений, который мы рассмотрим в этой части, — GIF.

Формат был разработан для передачи растровых изображений по интернету. GIF имеет цветовую палитру, состоящую из 256 цветов. Алгоритм GIF выбирает 256 наиболее используемых в исходном изображении цветов, а все остальные оттенки создаются путём подмешивания — подбора соседних пикселей таким образом, чтобы человеческий глаз воспринимал их как нужный цвет. По этой причине GIF не подходит для хранения полноцветных изображений и фотографий, но подходит для простейших анимаций.

Формат поддерживает прозрачность — каждый пиксель изображения может быть в двух состояниях: прозрачный или непрозрачный, полупрозрачность не поддерживается.

В последнее время GIF-изображения становятся всё менее используемыми и заменяются на другие, более оптимальные форматы.

Таким образом, формат GIF подходит если:

* нужна простейшая анимация.

# Размеры изображения

Мы узнали об основных форматах изображений, но Кекс опять прислал нам правки. При попытке прочитать нашу запись, какие-то картинки не поместились на экране его монитора, а какие-то было сложно разглядеть. Действительно, про размеры изображений мы подумать забыли.

В HTML, чтобы управлять шириной или высотой изображения, нужно использовать атрибуты width и height. Размеры в этих атрибутах задаются без единиц измерения px. Например:

<img src="logo.png" width="200" height="100">

В примере выше изображению задана ширина 200px и высота 100px.

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

Кстати, при работе с высотой и шириной изображения нужно быть очень аккуратным. Ведь если задать картинке одновременно и высоту, и ширину, то браузер может нарушить пропорции исходного изображения.

# Атрибут alt

Теперь картинки правильного размера, но Кекс спешит на бизнес-встречу в Москву и прямо сейчас едет в Сапсане. С интернетом произошли какие-то неполадки, и картинки не подгрузились — стало непонятно, что же на них изображено.

Специально для таких случаев, когда картинка не может отобразиться, и существует альтернативный текст. Ещё альтернативный текст помогает сайтам оставаться доступными, например, для категории пользователей, которая не имеет возможности видеть картинки. Что определённо является большим плюсом.

Альтернативный текст изображения задаётся с помощью атрибута alt. Например:

<img src="cat.png" alt="Кот, который гуляет сам по себе">

Теперь можно не бояться, что пользователь останется в неведении.

В общем, задавать альтернативный текст — хорошо.

Пробуем?

# Изображение-ссылка

В Москве после бизнеc-встречи Кекс понял, что на его сайте совсем нет никакого раздела «Обо мне». Он уже поставил задачу своим верстальщикам сделать такую страницу, а нам поручил сделать галерею со своими фотографиями.

Мы уже знаем, что существуют ссылки и картинки. А можем ли мы поместить одно в другое? Картинку в ссылку? Да. И сейчас узнаем как.

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 полноразмерную версию картинки или на отдельную страницу с этим изображением и подписью к нему.

Сейчас мы попробуем сделать в нашей галерее селфи ссылки-изображения, которые будут вести на отдельные странички с фотографиями.

# Теги figure и figcaption, демонстрационный материал

В галерее селфи каждая маленькая картинка-превьюшка ссылается на отдельную страницу с «фотокарточкой». Эта фотокарточка содержит полноразмерное изображение и его описание. Она достаточно независима, и позже мы сможем вставить её на любую другую страницу блога. Как правильно размечать такие «карточки»?

Для этого отлично подойдёт тег <figure>. Этот тег хорош для любого иллюстративного или демонстрационного материала, которым можно дополнить содержание документа. Таким материалом могут быть схемы, графики, примеры кода, таблицы и так далее.

Обычно каждый такой материал сопровождает разъясняющий комментарий, или «легенда». Для разметки этого комментария предназначен ещё один тег — <figcaption>, который размещается первым или последним элементом внутри <figure>. Например:

<figure>

*Схема, график, диаграмма или код*

<figcaption>Подпись к содержимому</figcaption>

</figure>

Давайте с помощью <figure> разметим фото в нашей галерее. Приступим.

# Ссылки с пустым href, атрибут title

На страницах с фотокарточками есть ссылки, которые ведут на предыдущую и следующую фотографии. Как быть с этой ссылкой, например, на последней фотокарточке? Ведь нет никакой следующей фотографии. Можно удалить ссылку целиком, а можно поступить иначе.

Тег <a> можно использовать вообще без адреса, то есть без атрибута href. Такой тег обозначает «ссылку-заглушку», которая в других условиях может стать обычной ссылкой (например, когда мы с последней фотки перейдём в середину галереи). Часто ссылки-заглушки используют, чтобы показать, что мы находимся на текущей странице:

<nav>

<ul>

<li><a>1 страница</a></li>

<li><a href="2">2 страница</a></li>

<li><a href="3">3 страница</a></li>

</ul>

</nav>

Когда мы удаляем атрибут href у ссылки, то лучше оставить подсказку о том, почему мы это сделали. Также подсказки помогают разъяснить назначение непонятных ссылок и ссылок-изображений. Подсказку можно добавить с помощью атрибута title. Например:

<a title="Ну какое назад? Вы на первой фотке!">Назад</a>

Подсказка появится, когда курсор задержится над ссылкой некоторое время.

Теперь давайте доработаем навигацию на последней странице галереи.

После выполнения этого шага можно смело скачивать [архив](https://htmlacademy.ru/assets/courses/305/project-state-3.zip?new) со сделанным нами Сайтом начинающего верстальщика.

# Конспект «Ссылки и изображения»

## Ссылки

### **Что такое ссылка**

Типичная ссылка представляет собой участок текста, щёлкая на который, вы переходите на другую страницу, открываете изображение, начинаете скачивать файл или перемещаетесь к какому-то месту на текущей странице.

Ссылки создаются с помощью тега <a>. Например:

<a href="https://htmlacademy.ru">HTML Academy</a>

Тег <a> можно использовать вообще без адреса, то есть без атрибута href. Такой тег обозначает «ссылку-заглушку», которая в других условиях может стать обычной ссылкой. Часто ссылки-заглушки используют, чтобы показать, что мы находимся на текущей странице:

<ul>

<li><a>1 страница</a></li>

<li><a href="2">2 страница</a></li>

<li><a href="3">3 страница</a></li>

</ul>

Когда мы удаляем атрибут href у ссылки, то лучше оставить подсказку о том, почему мы это сделали. Подсказку можно добавить с помощью атрибута title. Подсказка появится, когда курсор задержится над ссылкой некоторое время.

* 1 страница
* [2 страница](https://htmlacademy.ru/courses/305/run/16)
* [3 страница](https://htmlacademy.ru/courses/305/run/16)

### **Абсолютные ссылки**

Если нужно сделать ссылку на другой сайт в интернете, то необходимо использовать «обычный» адрес. Этот «обычный» или полный адрес называется абсолютным. Выглядит он, например, так:

https://site.ru/blog/index.html

Абсолютные адреса содержат минимум три части: протокол, имя сервера и путь.

### **Относительные ссылки**

Когда файл по ссылке должен открываться локально на компьютере, используются относительные адреса. В отличие от «обычных» адресов, в нём нет адреса сайта. Например:

day-2.html

Относительные адреса работают не только для файлов на компьютере, но и для страниц в сети. Если выложить два каких-то файла в интернет (не меняя их взаимное расположение), то их ссылка друг на друга всё равно будет работать.

Если в адресе нет имени сервера или протокола, то это относительный адрес.

### **Ссылки на файл**

По ссылкам можно не только переходить, но и скачивать файлы. Для этого необходимо просто в атрибуте href прописать ссылку на этот файл. А для того чтобы предотвратить открытие файлов прямо в браузере, у тега <a> существует атрибут download.

<a href="file.pdf" download>Браузер скачает меня, а не будет читать</a>

### **Ссылки-якоря**

Ссылка-якорь — это обычная ссылка, в адресе которой используется символ #, после которого следует идентификатор элемента. Идентификатор создаётся с помощью атрибута id у того тега, к которому надо перейти при щелчке по ссылке.

<a href="#part1">Глава 1</a>

Ссылка-якорь используется для прокрутки к заданной части страницы, в том числе используется и в абсолютных адресах.

## Изображения

### **Подключение изображений**

Для подключения изображений существует тег <img>, для него не требуется закрывающего тега, а путь к картинке задаётся в атрибуте src. Например:

<img src="logo.png">

### **Форматы изображений**

Существует несколько основных форматов изображений: JPEG, PNG, SVG и GIF.

Формат SVG переводится как масштабируемая векторная графика. Качество таких изображений не меняется при изменении размеров, да и вес у них небольшой. Отлично подходит для малоцветных схем, логотипов и иконок. Чаще всего используется в случаях, когда необходимо масштабировать изображение без потерь, изменять цвет элементов изображения, анимировать части изображения.

Формат JPEG подходит для фотографий, рисунков с большим количеством разноцветных деталей, изображений с плавным переходом яркости и контраста. При сжатии изображения ухудшается его качество.

Формат PNG позволяет сохранять изображения, в которых требуется особенная чёткость. Главная особенность этого формата — поддержка прозрачности. Подходит для изображений с прозрачностью и полупрозрачностью, когда необходима повышенная точность полноцветных изображений и для изображений с резкими переходами цветов.

Формат GIF используется для простейших анимаций. В последнее время GIF-изображения становятся всё менее используемыми и заменяются на другие, более оптимальные форматы.

### **Размеры изображения**

Для управления шириной или высотой изображения, используются атрибуты width и height. Размеры в этих атрибутах задаются без единиц измерения.

<img src="logo.png" width="200" height="100">

Если задать только один из размеров, ширину или высоту, то вторую размерность браузер вычислит самостоятельно исходя из пропорций изображения. Если задать картинке одновременно и высоту, и ширину, то браузер может нарушить пропорции исходного изображения.

### **Атрибут alt**

В случаях, когда картинка не может отобразиться, для того, чтобы было понятно, что на ней изображено, существует альтернативный текст. Также альтернативный текст помогает сайтам оставаться доступными, например, для категории пользователей, которая не имеет возможности видеть картинки.

Альтернативный текст изображения задаётся с помощью атрибута alt. Например:

<img src="cat.png" alt="Кот, который гуляет сам по себе">

### **Figure и figcaption**

Тег <figure> подходит для любого иллюстративного или демонстрационного материала, которым можно дополнить содержание документа: схемы, графики, примеры кода, таблицы и так далее. При удалении такого материала основное содержание не должно пострадать, иначе это не дополнительный материал и тег <figure> не подходит для его разметки.

Для разъясняющего комментария к такому иллюстративному материалу существует тег <figcaption>, который размещается первым или последним элементом внутри <figure>. Например:

<figure>

<img src="image.jpg" alt="Альтернативный текст">

<figcaption>Подпись к содержимому</figcaption>

</figure>

Подпись к содержимому

### **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>

# CSS-правила

Наполнение блога завершено, теперь пора его оформить. И для этого нам нужно подробнее узнать CSS.

CSS — это язык для оформления структурированных документов, например, HTML- документов. Синтаксис CSS незамысловат: это плоский список CSS-правил. CSS-правило состоит из селектора и перечня свойств и их значений:

селектор {

свойство: значение;

свойство: значение;

}

Ах да, комментарии в CSS тоже есть, для них используются символы /\* и \*/. Вот пример CSS-правила из первой части:

.feature-kitten {

padding-top: 60px; /\* отступ сверху \*/

background-image: url("img/bottle.svg"); /\* фоновая картинка \*/

}

Начнём оформление блока навигации. Пока просто раскомментируйте CSS-правило и посмотрите, что изменится.

Список CSS-правил мы называем «плоским», потому что в чистом CSS их нельзя вкладывать друг в друга. Возможно, в новых версиях языка эту возможность добавят.

Помимо CSS-правил (rule sets) в языке существуют так называемые «эт-правила» (at-rules), они начинаются с символа @, например, @font-face. «Эт-правила» ещё называют CSS-директивами.

Для решения наших текущих задач CSS-директивы не понадобятся, но мы обязательно познакомимся с ними позже.

# Селекторы

Селектор находится в начале CSS-правила, до фигурных скобок, и определяет к каким HTML-элементам применятся свойства и значения из правила. Вспомните пример:

.feature-kitten {

padding-top: 60px; /\* отступ сверху \*/

background-image: url("img/bottle.svg"); /\* фоновая картинка \*/

}

Строка .feature-kitten — это селектор. Она говорит браузеру применить список свойств (отступ и фон) ко всем элементам, у которых атрибут class имеет значение feature-kitten.

Простейшие (и самые популярные) селекторы — это селекторы по тегам и по классам. Селекторы по тегам содержат имя тега без символов < и > и применяются ко всем подходящим тегам. Селекторы по классам начинаются с точки, за которой идёт имя класса, и применяются ко всем тегам с подходящим атрибутом class. Например:

h1 { color: red; } /\* выберет все заголовки 1 уровня \*/

.info { color: blue; } /\* выберет только элементы с классом info \*/

Если у CSS-правил отличаются только селекторы, а свойства и значения одинаковые, то их можно сгруппировать через запятую. Например:

h1, .danger { color: red; }

/\* То же самое, что и \*/

h1 { color: red; }

.danger { color: red; }

А сейчас попробуйте использовать несколько разных типов селекторов. Подставляйте в CSS-правило селектор и смотрите, что будет меняться.

Селекторы очень разнообразны, мы посвятили им целых три части: [Селекторы. Знакомство](https://htmlacademy.ru/courses/42), [Селекторы. Погружение](https://htmlacademy.ru/courses/57) и [Селекторы. Тонкости](https://htmlacademy.ru/courses/58). Рекомендуем не спешить и пройти их после вводного тренажёра [«Знакомство с HTML и CSS»](https://htmlacademy.ru/courses/basic-html-css)

# Свойства и значения

Список свойств и значений находится внутри фигурных скобок CSS-правила. Пары «свойство-значение» отделяются друг от друга точкой с запятой, а свойство от значения отделяется двоеточием. Свойство определяет, какую характеристику внешнего вида мы хотим изменить, а значение — как именно. Ещё раз вспомните пример:

.feature-kitten {

padding-top: 60px; /\* отступ сверху \*/

background-image: url("img/bottle.svg"); /\* фоновая картинка \*/

}

В примере свойству padding-top задаётся значение 60px, а свойству background-image значение url("img/bottle.svg"). В итоге, это CSS-правило задаёт всем элементам с классом feature-kitten отступ сверху и фоновое изображение.

С помощью CSS можно задавать параметры отображения любого тега: ширину и высоту, отступы, цвет и размер шрифта, фон и так далее. Каждый раз, когда мы добавляем новое свойство или изменяем его значение, мы меняем что-то на странице. Давайте попробуем добавить несколько новых свойств и посмотрим, что изменится на странице блога.

# Наследование

На прошлом шаге мы задали белый цвет текста для nav, а он применился и для заголовка «Записи в блоге». Почему это произошло? Всё дело в наследовании.

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

Стили, присвоенные одному элементу, наследуются всеми потомками (вложенными элементами), но только в том случае, если они где-то явно не переопределены. Например, размер шрифта и его цвет достаточно применить к body, чтобы большинство элементов внутри имели те же свойства. Рассмотрим пример наследования:

body {

font-size: 14px;

}

nav {

font-size: 18px;

}

Размер шрифта у всего текста на странице, кроме текста внутри навигации, станет равен 14px. У nav есть своё *объявленное* значение размера шрифта (18px), и оно будет использоваться вместо *наследуемого* от body значения (14px). А ещё 18px станет новым наследуемым значением для потомков nav.

Если на странице из примера будут заголовки, то их размер тоже будет отличаться от 14px. Дело в том, что размер для заголовков тоже кое-где явно задан (и это кое-где мы обсудим в 11 задании). А значит у заголовков есть объявленное значение, которое будет использоваться вместо наследуемого значения.

Давайте убедимся, что наследование действительно работает. Смените название шрифта для всего body.

# Ненаследуемые свойства

В предыдущем задании мы перечислили основные наследуемые свойства. Но не все свойства наследуются. Основные ненаследуемые свойства — это параметры позиционирования, размеров, отступов, фона, рамок:

background, border, padding, margin, width, height, position и другие.

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.

Давайте посмотрим что было бы, если бы все подряд свойства наследовались. Вот так, например, выглядел бы nav на главной странице при добавлении ему рамки (свойства border):

## Записи в блоге

* [День первый. Как я забыл покормить кота](https://htmlacademy.ru/courses/307/run/6)
* [День второй. Хочу быть верстальщиком](https://htmlacademy.ru/courses/307/run/6)
* [День третий. Мой кот на меня обиделся](https://htmlacademy.ru/courses/307/run/6)

Теперь возьмём это же свойство border и убедимся, что оно не наследуется.

# Составные свойства

В CSS есть обычные свойства, управляющие одним параметром отображения, и есть составные свойства, управляющие одновременно несколькими параметрами.

Например, свойство font-size — обычное, оно управляет только размером шрифта. А свойство font — составное, оно задаёт сразу шесть параметров: размер и название шрифта, высоту строки и некоторые другие. Браузер всегда «расшифровывает» составные свойства в обычные. Например, такое составное свойство:

font: 16px/26px "Arial", sans-serif;

Браузер «расшифрует» в такой набор обычных свойств и их значений:

font-size: 16px; /\* было задано в font \*/

line-height: 26px; /\* было задано в font \*/

font-family: "Arial", sans-serif; /\* было задано в font \*/

*font-weight: normal;* /\* не было задано в font \*/

*font-style: normal;* /\* не было задано в font \*/

*font-variant: normal;* /\* не было задано в font \*/

Если значение обычного свойства не было задано в составном, то браузер при «расшифровке» использует *исходное значение* этого свойства. В примере значение 16px для font-size взято из font, а для font-weight использовано исходное значение — normal.

Составное свойство всегда задаёт значения всем своим компонентам. Для не заданных явно компонентов используются исходные значения. Поэтому составные свойства нужно использовать с осторожностью. Например, если забыть описать высоту строки:

font: 16px "Arial", sans-serif;

То для line-height браузер возьмёт исходное значение, и внешний вид текста может оказаться плохим.

Пора добавить странице блога красивый фон. Используем для этого составное свойство background.

# Типы значений: абсолютные и относительные

Давайте сделаем аватарку модной, то есть круглой. Можно увеличить значение border-radius до половины размера аватарки, то есть до 40px, и получить круг. Но если потом появится новая аватарка с другими размерами, то снова придётся менять стили. Можно ли сразу сделать универсально? Да! С помощью процентов:

border-radius: 50%;

Если радиус скругления задать в процентах, то он будет рассчитываться относительно размеров элемента и будет изменяться вместе с этими размерами. Если этот радиус задать в пикселях, то он абсолютно всегда будет одинаковым. Вот и единицы измерения в CSS делятся на *абсолютные* и *относительные*.

Абсолютные единицы измерения привязаны к настоящим физическим размерам и связаны между собой жёсткими пропорциями. Примеры абсолютных единиц измерения:

font-size: 1cm; /\* 1 сантиметр \*/

font-size: 10mm; /\* В 1 сантиметре 10 миллиметров \*/

font-size: 38px; /\* В 1 сантиметре 38 пикселей \*/

Пиксели, px, используют чаще всего, остальные абсолютные единицы почти не применяют.

Относительные единицы измерения описывают значения, которые зависят от других значений. Например, ширина элемента в процентах зависит от ширины родительского элемента, а ширина элемента в em зависит от размера шрифта самого элемента.

К относительным единицам относятся em, rem, vh, vw и некоторые другие, ну и, конечно же, проценты. Каждая из таких единиц решает свой круг задач. Например, проценты используют для «резиновой» вёрстки, а em применяют в вёрстке государственных сайтов с особыми дополнительными требованиями к масштабированию текста.

Перечень всех [абсолютных единиц измерения](https://www.w3.org/TR/css3-values/#absolute-lengths) и их соотношений есть в спецификации. Там же, в спецификации, есть перечень всех [относительных единиц измерения](https://www.w3.org/TR/css3-values/#relative-lengths) и описание правил расчёта.

# Селекторы по тегам и по классам

Помните, на втором шаге мы познакомились с селекторами по тегам и по классам? И даже попробовали стилизовать один из тегов — nav. Наш проект растёт, количество стилей увеличивается, поэтому пришло время разобраться, как лучше использовать эти селекторы, чтобы избежать ошибок в будущем.

Представьте, что вам нужно застилизовать список, ul. Казалось бы, можно сделать вот так и дело с концом:

ul {

свойство: значение;

}

Но вот незадача, на странице может быть несколько списков, и стили применятся ко всем спискам, даже к тем, которые вы менять не хотели. Чтобы избежать таких ситуаций, лучше не использовать селекторы по тегам или использовать их как можно реже.

Нашу проблему отлично решает использование селектора по классу. Добавив нужный класс к элементу разметки, мы будем уверены, что стили применятся именно к этому элементу и ни к какому больше.

Давайте проведём небольшое улучшение кода и в CSS-правиле для навигации заменим селектор по тегу на селектор по классу.

# Вложенные селекторы

Если селекторы по тегам нужно использовать пореже, а по классам почаще, то придётся каждому тегу в разметке добавлять класс? Нет, это ненужное усложнение кода.

Зачем мы использовали классы? Чтобы ограничить количество тегов, к которым применится CSS-правило: не ко всем тегам, а к тегам с нужным классом. Но ограничивать область применения стилей можно и по-другому. Вот список внутри навигации:

<nav class="blog-navigation">

<ul>…</ul>

</nav>

Как применить стили только к этому списку? Первый способ: добавить ему класс и использовать селектор по этому классу. Второй способ: использовать специальный селектор, чтобы стили применялись к спискам внутри блока с классом blog-navigation.

Вы можете комбинировать любые типы селекторов через пробел. Такие селекторы называются *вложенными* или *контекстными* и читаются справа налево. Например:

nav a {…} /\* выберет теги a внутри тегов nav \*/

.menu ul {…} /\* теги ul внутри тегов с классом menu \*/

.post .title {…} /\* теги с классом title внутри тегов с классом post \*/

Узнали селектор nav a из [второго задания](https://htmlacademy.ru/courses/307/run/2)? Он помог сделать ссылки белыми только в навигации, в остальном тексте они остались синими.

Комбинировать можно любое количество селекторов, но лучше использовать двойную или максимум тройную вложенность. Вложенные селекторы спасают нас от необходимости придумывать имена классов и загромождать ими разметку.

Давайте изменим пару селекторов в стилях блога, а заодно улучшим оформление заголовков и элементов списка внутри навигации (уже используя правильные селекторы

# Стили по умолчанию

Некоторым элементам можно не задавать никаких стилей, но у них всё равно будет какое-то оформление. Например, списки «без стилей» выглядят так:

* первый пункт списка,
* второй пункт списка,
* третий пункт списка.

У списка есть отступы и маркеры, но откуда они берутся? Потому что список такой «сам по себе»? Нет! Параметры оформления тегов описываются только в CSS, и наши списки — не исключение. Значит где-то есть стили, в которых спискам заданы отступы и маркеры? Да! И эти стили хранятся внутри браузера, это браузерные стили по умолчанию.

Помните, как в [4 задании](https://htmlacademy.ru/courses/307/run/4) мы говорили, что размер шрифта body не будет наследоваться заголовками? Это происходит из-за того, что размер шрифта заголовков явно задан внутри браузерных стилей по умолчанию (и наследуемое от body значение им не нужно).

Ваши навыки существенно подросли! Пора сделать раздел про них более подробным. Как всегда начнём с разметки. Для описания уровня навыков лучше всего подойдёт список определений. Заодно посмотрите, что у тегов dl/dt/dd тоже есть стили по умолчанию.

# Каскадирование

Вот мы и подобрались к одному из самых важных механизмов CSS — каскадности. Именно он скрывается за первой С в аббревиатуре CSS (Cascading Style Sheets).

Когда браузер отрисовывает страницу, он должен определить итоговый вид каждого HTML-элемента. Для этого он собирает все CSS-правила, которые относятся к каждому элементу, ведь на элемент могут влиять сразу несколько CSS-правил. Например:

<p class="beloved-color">Зелёный - мой любимый цвет</p>

На этот элемент могут одновременно влиять CSS-правила по тегу p и по классу beloved-color из наших стилей, да ещё и CSS-правило по тегу p из браузерных стилей.

После того, как все правила для элемента собраны, браузер комбинирует все свойства из этих правил и применяет их к элементу. Если в наших стилях есть такой код:

p { font-size: 14px; }

.beloved-color { color: green; }

То у нашего абзаца про цвет будет такой итоговый набор свойств и значений:

font-size: 14px; /\* из правила для p в наших стилях \*/

color: green; /\* из правила для .beloved-color в наших стилях \*/

margin: 1em 0; /\* из правила для p в браузерных стилях \*/

Этот механизм комбинирования стилей из разных источников в итоговый набор свойств и значений для каждого HTML-элемента и называется каскадностью.

Давайте продолжим оформление списка навыков, а заодно применим каскад.

На один элемент могут действовать несколько CSS-правил. Если в этих правилах есть одинаковые свойства с разными значениями, то возникает конфликт. Например:

ul { list-style: disc; } /\* браузерные стили \*/

.blog-navigation ul { list-style: none; } /\* наши стили \*/

На один и тот же список в нашем блоге действуют стили с разными значениями свойства list-style. Это и есть конфликт, ведь у свойства может быть только одно значение.

Браузеру нужно как-то решать, какими будут итоговые значения конфликтующих свойств. Конфликт разрешается максимум за три шага. Если на текущем шаге определиться не удалось, то выполняется следующий шаг. Вот эти шаги:

1. Сравниваются приоритеты стилевых файлов, в которых находятся конфликтующие свойства. Например, *авторские* (то есть наши) стили приоритетнее браузерных.
2. Сравнивается специфичность селекторов у CSS-правил с конфликтующими свойствами. Например, селектор по классу более специфичен, чем селектор по тегу.
3. Побеждает то свойство, которое находится ниже в коде.

Мы подробно разберём приоритеты, специфичность селекторов и прочие тонкости наследования и каскадирования в [соответствующей части](https://htmlacademy.ru/courses/66). А пока вспомните margin: 0; для .skills dd на прошлом шаге. Свойство из наших стилей (более приоритетных) вступило в конфликт со свойством из браузерных стилей и победило, обнулив отступы.

Другой конфликт возник в правиле .skills dd при появлении там margin-bottom:

margin: 0;

margin-bottom: 10px;

Каскад работает и внутри CSS-правил, поэтому в конфликт вступили «обычный» margin-bottom и аналогичный компонент составного свойства. «Обычное» свойство победило, так как находится ниже в коде:

*margin-left: 0;* /\* из составного свойства \*/

*margin-top: 0;* /\* из составного свойства \*/

*margin-right: 0;* /\* из составного свойства \*/

/\* из составного свойства \*/

margin-bottom: 10px;

А сейчас давайте превратим проценты освоения навыков в шкалы прогресса. Для этого придётся добавить в разметку дополнительные декоративные обёртки, и лучше всего для этого подходит [уже знакомый](https://htmlacademy.ru/courses/301/run/19) тег <div>. Исходные стили для шкал уже подготовлены.

# Конфликт свойств

На один элемент могут действовать несколько CSS-правил. Если в этих правилах есть одинаковые свойства с разными значениями, то возникает конфликт. Например:

ul { list-style: disc; } /\* браузерные стили \*/

.blog-navigation ul { list-style: none; } /\* наши стили \*/

На один и тот же список в нашем блоге действуют стили с разными значениями свойства list-style. Это и есть конфликт, ведь у свойства может быть только одно значение.

Браузеру нужно как-то решать, какими будут итоговые значения конфликтующих свойств. Конфликт разрешается максимум за три шага. Если на текущем шаге определиться не удалось, то выполняется следующий шаг. Вот эти шаги:

1. Сравниваются приоритеты стилевых файлов, в которых находятся конфликтующие свойства. Например, *авторские* (то есть наши) стили приоритетнее браузерных.
2. Сравнивается специфичность селекторов у CSS-правил с конфликтующими свойствами. Например, селектор по классу более специфичен, чем селектор по тегу.
3. Побеждает то свойство, которое находится ниже в коде.

Мы подробно разберём приоритеты, специфичность селекторов и прочие тонкости наследования и каскадирования в [соответствующей части](https://htmlacademy.ru/courses/66). А пока вспомните margin: 0; для .skills dd на прошлом шаге. Свойство из наших стилей (более приоритетных) вступило в конфликт со свойством из браузерных стилей и победило, обнулив отступы.

Другой конфликт возник в правиле .skills dd при появлении там margin-bottom:

margin: 0;

margin-bottom: 10px;

Каскад работает и внутри CSS-правил, поэтому в конфликт вступили «обычный» margin-bottom и аналогичный компонент составного свойства. «Обычное» свойство победило, так как находится ниже в коде:

*margin-left: 0;* /\* из составного свойства \*/

*margin-top: 0;* /\* из составного свойства \*/

*margin-right: 0;* /\* из составного свойства \*/

/\* из составного свойства \*/

margin-bottom: 10px;

А сейчас давайте превратим проценты освоения навыков в шкалы прогресса. Для этого придётся добавить в разметку дополнительные декоративные обёртки, и лучше всего для этого подходит [уже знакомый](https://htmlacademy.ru/courses/301/run/19) тег <div>. Исходные стили для шкал уже подготовлены.

# Множественные классы

Давайте закрепим приём стилизации с помощью множественных классов, который мы уже использовали в [первой части стартового тренажёра](https://htmlacademy.ru/courses/297/run/10). Чтобы вспомнить этот приём, разберём небольшой пример. Допустим, у вас на сайте есть разные типы уведомлений:

<div class="alert">Прост сообщение.</div>

<div class="alert alert-error">Оши-ы-ы-бка! Всё пропа-а-а-ло!</div>

У них все стили кроме фона одинаковые. Фон обычных сообщений серый, фон ошибок красный. Стили для этих уведомлений можно организовать так:

.alert {

*/\* свойства для рамок, отступов и так далее \*/*

background-color: lightgrey;

}

.alert-error { background-color: red; }

Класс alert с общими стилями есть у всех сообщений. Класс alert-error с частными стилями есть только у ошибок. **Помните:** несколько классов в атрибуте class задаются через пробел.

Почему код лучше организовать именно так? Мы уже знаем, как работает каскад в CSS, поэтому ответить будет легко.

Использование нескольких классов в разметке помогает явно указать, какие стили будут смешиваться с помощью каскада. И, конечно, эти стили удобнее всего разместить в коде рядом друг с другом.

Использование однотипных селекторов (например, по классам) позволяет использовать самый простой механизм разрешения конфликтов — по порядку в коде. Размещаем частные CSS-правила после общего, и всё что нужно точно переопределится.

А теперь уже с полным пониманием происходящего давайте улучшим блок навыков: сделаем шкалу с высоким процентом зелёной. Для этого в разметку добавим ещё один класс, а в стили — CSS-правило для этого класса, причём после общего CSS-правила.

# Встроенные стили, атрибут style

Мы уже знаем, что хорошим тоном считается подключать стили через тег <link>:

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

Но есть ещё один способ подключения стилей — встраивание прямо в документ вместо подключения по ссылке. Стили подключатся либо внутри тега <style>, либо прописываются в значении атрибута style самих HTML-элементов.

Вариант подключения стилей в теге <style> используется чаще всего для оптимизации загрузки страницы, ведь в таком случае браузер не будет отправлять дополнительных запросов на сервер. Тег <style> обычно размещают внутри <head>. Например:

<head>

<style>

CSS-код

</style>

</head>

Второй вариант встраивания стилей — содержимое атрибута style. Свойства и значения, прописанные таким образом, применятся точечно к одному элементу:

<div style="width: 50%;"></div>

Обычно использование этого способа считается плохой практикой. Но иногда в виде исключения бывает удобнее воспользоваться встраиванием стилей в атрибут style, чем писать отдельные CSS-правила. Например, когда нужно управлять стилями именно из разметки, и создавать отдельные классы при этом будет излишне. Так бывает, когда какие-то стилевые параметры устанавливаются с помощью сторонних программ или другими людьми, например, через CMS.

Давайте подробнее разберём пример со style="width: 50%;". Предположим, что нужно иметь возможность управлять в разметке шириной с помощью свойства width в диапазоне от 0% до 100%. Чтобы сделать это через CSS, пришлось бы создать 100 классов, и применять их, например:

.width-0 { width: 0% }

.width-1 { width: 1% }

.width-2 { width: 2% }

/\* и так далее… \*/

Намного удобнее будет сделать точечное встраивание стиля с шириной в атрибут style.

Давайте и воспользуемся этим приёмом для стилизации прогресса навыков.

# Конспект: основы CSS

## CSS-правила

CSS — это язык для оформления структурированных документов, например, HTML- документов. Синтаксис — это плоский список CSS-правил. CSS-правило состоит из селектора и перечня свойств и их значений:

селектор {

свойство: значение;

свойство: значение;

}

Для комментариев в CSS используются символы /\* и \*/.

## Селекторы

Селектор находится в начале CSS-правила, до фигурных скобок, и определяет, к каким HTML-элементам применятся свойства и значения из правила.

.feature-kitten {

padding-top: 60px;

}

Простейшие (и самые популярные) селекторы — это селекторы по тегам и по классам. Селекторы по тегам содержат имя тега без символов < и > и применяются ко всем подходящим тегам. Селекторы по классам начинаются с точки, за которой идёт имя класса, и применяются ко всем тегам с подходящим атрибутом class.

h1 { color: red; }

.info { color: blue; }

На странице может быть несколько списков, и стили применятся ко всем спискам, даже к тем, которые вы менять не хотели. Чтобы избежать таких ситуаций, лучше не использовать селекторы по тегам или использовать их как можно реже.

Если у CSS-правил отличаются только селекторы, а свойства и значения одинаковые, то их можно сгруппировать через запятую.

Также можно комбинировать любые типы селекторов через пробел. Такие селекторы называются вложенными или контекстными и читаются справа налево. Например:

nav a {…}

.menu ul {…}

.post .title {…}

## Свойства и значения

Список свойств и значений находится внутри фигурных скобок CSS-правила. Свойство определяет, какую характеристику внешнего вида мы хотим изменить, а значение — как именно.

.feature-kitten {

padding-top: 60px;

}

Каждый раз, когда мы добавляем новое свойство или изменяем его значение, мы меняем что-то на странице.

## Наследование

Наследование в CSS — это механизм, с помощью которого значения свойств элемента-родителя передаются его элементам-потомкам. Стили, присвоенные одному элементу, наследуются всеми потомками (вложенными элементами), но только в том случае, если они где-то явно не переопределены.

## Составные свойства

В CSS есть обычные свойства, управляющие одним параметром отображения, и есть составные свойства, управляющие одновременно несколькими параметрами. Например, свойство font. Оно задаёт сразу шесть параметров: размер и название шрифта, высоту строки и некоторые другие.

font: 16px/26px "Arial", sans-serif;

Если значение обычного свойства не было задано в составном, то браузер при «расшифровке» использует исходное значение этого свойства.

## Типы значений: абсолютные и относительные

Абсолютные единицы измерения привязаны к настоящим физическим размерам и связаны между собой жёсткими пропорциями. Пиксели, px, используют чаще всего, остальные абсолютные единицы почти не применяют. Примеры абсолютных единиц измерения:

font-size: 1cm;

font-size: 10mm;

font-size: 38px;

Относительные единицы измерения описывают значения, которые зависят от других значений. Например, ширина элемента в процентах зависит от ширины родительского элемента, а ширина элемента в em зависит от размера шрифта самого элемента. К относительным единицам относятся em, rem, vh, vw и некоторые другие, ну и, конечно же, проценты.

## Стили по умолчанию

Некоторым элементам можно не задавать никаких стилей, но у них всё равно будет какое-то оформление. Например, у списка <ul> есть отступы и маркеры. Такие стили называются стилями по умолчанию и задаются внутри браузерных стилей изначально. Их можно переопределить или сбросить, задав другие значения свойств элементу.

## Каскадирование

Когда браузер отрисовывает страницу, он должен определить итоговый вид каждого HTML-элемента. Для этого он собирает все CSS-правила, которые относятся к каждому элементу, ведь на элемент могут влиять сразу несколько CSS-правил. Механизм комбинирования стилей из разных источников в итоговый набор свойств и значений для каждого тега называется каскадностью. Например, есть такой элемент в разметке:

<p class="beloved-color">Зелёный - мой любимый цвет</p>

Заданные стили:

.beloved-color { color: green; }

Браузерные стили:

margin: 1em 0;

Итоговые стили:

color: green;

margin: 1em 0;

## Конфликт свойств

На один элемент могут действовать несколько CSS-правил. Если в этих правилах есть одинаковые свойства с разными значениями, то возникает конфликт. Например:

ul { list-style: disc; }

.blog-navigation ul { list-style: none; }

Браузеру нужно как-то решать, какими будут итоговые значения конфликтующих свойств. Конфликт разрешается максимум за три шага. Если на текущем шаге определиться не удалось, то выполняется следующий шаг. Вот эти шаги:

1. Сравниваются приоритеты стилевых файлов, в которых находятся конфликтующие свойства. Например, авторские (то есть наши) стили приоритетнее браузерных.
2. Сравнивается специфичность селекторов у CSS-правил с конфликтующими свойствами. Например, селектор по классу более специфичен, чем селектор по тегу.
3. Побеждает то свойство, которое находится ниже в коде.

Каскад работает и внутри CSS-правил.

## Встраивание и подключение внешних стилей

Внешние стили подключаются через тег <link>

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

Встраивание стилей в тег <style>. Его обычно размещают внутри <head>:

<head>

<style>

CSS-код

</style>

</head>

Такой способ используется для оптимизации загрузки страницы, ведь в таком случае браузер не будет отправлять дополнительных запросов на сервер.

Встраивание в атрибут style:

<div style="width: 50%;"></div>

Свойства и значения, прописанные таким образом, применятся точечно к одному элементу.

Обычно использование этого способа считается плохой практикой. Но иногда в виде исключения бывает удобнее воспользоваться встраиванием стилей в атрибут style, чем писать отдельные CSS-правила. Например, когда нужно управлять стилями именно из разметки, и создавать отдельные классы при этом будет излишне. Так бывает, когда какие-то стилевые параметры устанавливаются с помощью сторонних программ или другими людьми, например, через CMS

# Свойство font-size, размер шрифта

В [прошлой части](https://htmlacademy.ru/courses/307) мы познакомились с основными механизмами CSS, но не рассматривали подробности работы отдельных CSS-свойств. Поэтому в этой части вас ожидает полное погружение в CSS. А начнём его с оформления основы любой страницы в интернете — а именно с оформления текста.

Самый главный «текстовый» параметр, которым можно управлять — это шрифт. Какой именно задан шрифт, какого он размера, начертания и насыщенности — разберём эти свойства по порядку.

Размером шрифта управляет свойство font-size. Значение свойства задаёт желаемую высоту символа шрифта. Причём единицы измерения могут быть абсолютными или относительными. Мы уже рассматривали единицы измерения в одном из [заданий прошлой части](https://htmlacademy.ru/courses/307/run/8), при работе с размерами шрифта они также актуальны.

Самая часто используемая единица измерения размера шрифта — пиксели px:

p {

font-size: 20px;

}

На самом деле абсолютную величину размера шрифта можно задавать не только в пикселях, пунктах и других величинах, но и особыми «шрифтовыми» ключевыми словами: xx-small, x-small, small, medium, large, x-large и xx-large. На практике этими единицами измерения пользуются редко. Подробно об этих ключевых словах можно узнать в [спецификации](https://www.w3.org/TR/css-fonts-3/#font-size-prop).

# Относительный размер шрифта

Абсолютные величины применительно к размеру шрифта — это простое, но не совсем гибкое решение и применяется для самых простых случаев.

К примеру, для всего документа задан размер шрифта:

body { font-size: 16px; }

В [задании про наследование](https://htmlacademy.ru/courses/307/run/4) мы разбирали, что все дочерние элементы с *необъявленным* значением размера шрифта унаследуют этот размер, то есть 16px.

Заголовку на странице мы зададим другое фиксированное значение размера шрифта:

h1 { font-size: 32px; }

Пока что всё хорошо. Но представьте, что теперь появляется требование: на больших экранах мониторов увеличить размер шрифта документа с 16px до 20px. Если поменять размер у body, то у «наследников» размер тоже изменится. Но у заголовка размер никак не поменяется, он будет фиксированным — 32px.

Хочется сделать так, чтобы при изменении основного размера шрифта для родителя (в нашем случае это body), его дочерние элементы пропорционально меняли свои размеры шрифта. И для этого случая есть специальная единица измерения — em.

Величина 1em — это *такой же* размер шрифта, что и у родителя. Соответственно, если мы хотим, чтобы шрифт дочернего элемента был всегда в 2 раза больше родительского, то зададим значение 2em:

h1 { font-size: 2em; }

Такой подход к написанию стилей позволяет сделать код более гибким.

Наравне с «абсолютными» ключевыми словами существует пара «относительных» ключевых слов, которые применяются к размеру шрифта: larger и smaller. Они буквально делают размер шрифта элемента *больше* или *меньше* того размера, который задан его родительскому элементу. Подробно об этих ключевых словах можно узнать в [спецификации](https://www.w3.org/TR/css-fonts-3/#relative-size-value).

# Свойство line-height, высота строки

Окей, с помощью свойства font-size мы управляем размером отдельных символов шрифта. Но дело в том, что тексты — это обычно больше, чем набор символов. Тексты состоят из строк, которые располагаются друг под другом. И хотелось бы уметь управлять расположением этих строк относительно друг друга и другого контента.

Для этого в CSS есть свойство line-height. Оно управляет высотой строки или, если точнее, межстрочным интервалом.

Как и в случае с размером шрифта, значения line-height можно задавать разными способами.

По умолчанию это свойство имеет значение normal. Оно указывает браузеру, что межстрочный интервал нужно подобрать автоматически, исходя из размера шрифта. Спецификация рекомендует устанавливать его в пределах 100-120% от размера шрифта. То есть:

p {

font-size: 10px;

line-height: normal; /\* значение будет примерно 12px \*/

}

Значение normal позволяет всем нестилизованным текстам выглядеть удобочитаемо. Однако, если есть необходимость отойти от стилизации по умолчанию, line-height можно задать фиксированное абсолютное значение в px.

p {

font-size: 16px;

line-height: 26px;

}

**Относительная высота строки**

Итак, какие варианты для line-height мы уже рассмотрели:

* относительное значение normal, вычисляется браузером в зависимости от font-size автоматически, *мы не можем им управлять*;
* абсолютное значение в px.

А если нам нужно задать относительное значение, но не такое, как normal? Так тоже можно. Для этого значение line-height задаётся в процентах или в виде множителя. В таком случае браузер вычисляет значение динамически в зависимости от font-size:

p {

font-size: 10px;

line-height: 150%; /\* вычисленное значение: 10px \* 150% = 15px \*/

line-height: 2; /\* вычисленное значение: 10px \* 2 = 20px \*/

}

Как вы понимаете, относительные значения более гибкие, чем абсолютные. Но для простых сайтов, таких как наш, фиксированных font-size и line-height будет вполне достаточно.

**Свойство font-family, семейство шрифта**

Мы уже разобрались с размерами шрифта, но как же задавать сам шрифт?

Для этого есть специальное свойство — font-family. В значении этого свойства указывается список названий шрифтов, перечисленных через запятую. В начале списка располагают самый желаемый шрифт, затем менее желаемый, а в самом конце списка — общий тип шрифта. Браузер проходит по списку слева направо и использует первый найденный в системе или на сайте шрифт. Пример:

body {

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

}

Если название шрифта состоит из нескольких слов, то его нужно заключать в кавычки. Одинарные названия можно оставлять как без кавычек, так и с ними для единообразия. Если браузер не находит на сайте или в операционной системе ни одно из перечисленных названий шрифтов, то он применяет последнее значение — общий системный тип шрифта. Самые распространённые типы шрифта:

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

На самом деле, кроме serif и sans-serif есть ещё менее распространённые типы шрифта:

* monospace — моноширинный шрифт;
* cursive — шрифт с неформальным начертанием, например, имитация рукописного текста или леттеринга;
* fantasy — декоративный шрифт, например, всемирно известный Comic Sans.

**Свойство font-weight, насыщенность шрифта**

Ещё один параметр шрифта, которым мы можем управлять — это насыщенность или толщина шрифта. Шрифт может быть жирнее или тоньше обычного начертания. Устанавливается насыщенность с помощью свойства font-weight. В качестве значения можно использовать ключевое слово или число. Самые часто встречающиеся значения:

* 400 или normal — обычный шрифт, значение по умолчанию;
* 700 или bold — жирный шрифт.

Например:

h1 {

font-weight: 400; /\* то же самое что и normal \*/

}

p {

font-weight: bold; /\* то же самое что и 700 \*/

}

На самом деле, font-weight может принимать одно из девяти числовых вариантов насыщенности:

* 100: Thin;
* 200: Extra Light (Ultra Light);
* 300: Light;
* 400: Normal;
* 500: Medium;
* 600: Semi Bold (Demi Bold);
* 700: Bold;
* 800: Extra Bold (Ultra Bold);
* 900: Black (Heavy).

Все эти числовые значения задают степень толщины шрифта от самого тонкого до самого толстого.

Но в большинстве системных шрифтов всё равно есть только два варианта толщины: обычный normal (400) и жирный bold (700). Поэтому и остальные значения свойства используются реже.

Кроме перечисленных выше числовых значений у font-weight может быть ещё два относительных значения: bolder и lighter. Эти значения делают шрифт жирнее и тоньше, чем текущее или унаследованное значение.

На самом деле, font-weight может принимать одно из девяти числовых вариантов насыщенности:

* 100: Thin;
* 200: Extra Light (Ultra Light);
* 300: Light;
* 400: Normal;
* 500: Medium;
* 600: Semi Bold (Demi Bold);
* 700: Bold;
* 800: Extra Bold (Ultra Bold);
* 900: Black (Heavy).

Все эти числовые значения задают степень толщины шрифта от самого тонкого до самого толстого.

Но в большинстве системных шрифтов всё равно есть только два варианта толщины: обычный normal (400) и жирный bold (700). Поэтому и остальные значения свойства используются реже.

Кроме перечисленных выше числовых значений у font-weight может быть ещё два относительных значения: bolder и lighter. Эти значения делают шрифт жирнее и тоньше, чем текущее или унаследованное значение.

# Свойство text-align, горизонтальное выравнивание текста

Следующее свойство text-align описывает, как выравнивается текст внутри блока по горизонтали (на самом деле не только текст, но об этом в [следующем задании](https://htmlacademy.ru/courses/309/run/8)).

Свойство может принимать следующие значения:

1. left — выравнивание по левому краю блока, это значение по умолчанию;
2. right — по правому краю блока;
3. center — по центру блока;
4. justify — по ширине блока, при этом слова в строке будут размещаться так, чтобы занять равномерно всё пространство строки (пробелы между словами в таком случае становятся неравномерными, так как браузер «растягивает» слова в строке).

Важно помнить, что свойство text-align применяется именно к самому блоку-контейнеру, внутри которого находится текстовый контент:

HTML:

<p>

Я текст внутри абзаца

</p>

CSS:

p {

text-align: center;

}

На самом деле существует ещё два значения text-align: это выравнивание к началу блока start и выравнивание к концу блока end. Если вы попробуете применить эти значения, то увидите, что эффект от значения start совпадает со значением left, а end делает то же самое, что и right.

Но это не всегда так. Дело в том, что не во всех языках текст пишется слева направо. Например, в японском текст пишется сверху вниз, и в этом случае значения left и right уже не так логичны, как в «европейских» языках.

Именно чтобы сделать CSS-код универсальным для любых направлений написания текста и появились такие значения text-align. Кстати, для управления направлением текста есть специальное свойство — [writing-mode](https://www.w3.org/TR/css-writing-modes-3/#propdef-writing-mode).

# Выравнивание text-align не только для текста

[На прошлом шаге](https://htmlacademy.ru/courses/309/run/7) мы упоминали, что свойство text-align способно влиять не только на тексты, но и другие элементы. Эти элементы должны быть *инлайновыми*, то есть это изображения, инлайн-блоки, инлайн-таблицы и другие.

Например, сейчас в задании мы разберём такой приём, который больше напоминает трюк, так как мы используем text-align для выравнивания картинки <img> внутри контейнера, то есть по факту для создания простейшей «сетки». Но об этой особенности свойства text-align очень полезно знать, так как это вполне легальный способ выравнивания элементов.

Вперёд к заданию.

Тема инлайнового контекста форматирования — целый отдельный мир CSS. В [спецификации](https://www.w3.org/TR/css-display-3/#inline-level) описаны разные виды блоков, которые участвуют в формировании потока документа. Также есть [замечательная книга](https://css-live.ru/articles/obzor-inlajnovyj-kontekst-formatirovaniya.html) на русском об этой простой на первый взгляд, но на самом деле достаточно комплексной теме.

**Свойство vertical-align, вертикальное выравнивание текста**

Если есть свойство для горизонтального выравнивания, то должно быть и для вертикального? Да, и это свойство vertical-align.

Этим свойством можно выравнивать *инлайновые* элементы относительно содержащей его строки. Самый простой пример, который мы разберём в этом задании — выровнять картинку <img> по вертикали в текстовой строке.

У свойства vertical-align много значений, но самые часто используемые:

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

В отличие от text-align свойство vertical-align задаётся самому элементу, а не содержащему его контейнеру:

HTML:

<p>

<img src="picture.png" alt="Я картинка">

Я текст внутри абзаца

</p>

CSS:

img {

vertical-align: middle;

}

Помимо ключевых слов для vertical-align ещё есть возможность задавать значения в пикселях и процентах.

Процентное значение (например, vertical-align: 100%) вычисляется относительно заданной высоты строки line-height. Значение 0% — это то же самое, что и baseline, а ещё оно может быть как положительным, так и отрицательным.

Значение в пикселях (например, vertical-align: 10px) работает аналогично процентному, только отсчитывается в пикселях в большую или меньшую сторону. Значение 0px эквивалентно baseline.

# Использование фона для оформления текста

Ещё один важный параметр текста — его цвет. Управляется он свойством color. Подробно это свойство и его значения разберём в [следующем задании](https://htmlacademy.ru/courses/309/run/11). А в этом узнаем, как можно использовать фон блока для оформления текста, а также как цвет фона и цвет текста должны сочетаться.

Цвет в CSS чаще всего задаётся в шестнадцатеричном формате с символом решётки в начале:

#000000 — чёрный цвет;

#ffffff — белый цвет;

#ff0000 — красный цвет и так далее.

Фоновое изображение для блока задаётся свойством background-image, а фоновый цвет — свойством background-color:

div {

background-image: url("путь\_к\_файлу\_изображения.png");

background-color: #cccccc;

}

Фоновое изображение и фоновый цвет блока всегда должны достаточно сильно контрастировать с цветом текста. Чем больше контраст, тем удобнее читается текст в разных условиях освещённости и на разных устройствах. Поэтому если вы задаёте блоку фоновое изображение, нужно обязательно дополнительно задавать подходящий фоновый цвет. В этом случае, пока изображение загружается, или в случае, если оно совсем не загрузится, текст всё равно можно будет прочитать:

p {

/\* идеальный контраст: цвет текста белый, цвет фона — чёрный \*/

background-color: #000000;

color: #ffffff;

}

span {

/\* плохой контраст: цвет текста и фона — серые \*/

background-color: #cccccc;

color: #dddddd;

}

Подробно работа с фонами в CSS рассматривается в части «[Фоны. Знакомство](https://htmlacademy.ru/courses/53)» и «[Рамки и фоны. Погружение](https://htmlacademy.ru/courses/88)».

# Свойство color, цвет текста

Мы помним, что цветом текста и фона можно управлять свойствами color и background-color.

Теперь разберёмся подробно со значениями этих свойств.

Цвет может быть задан в виде ключевого слова (полный список ключевых слов приводится в [спецификации](https://www.w3.org/TR/css-color-3/#svg-color)). Например:

color: black; /\* чёрный цвет \*/

color: red; /\* красный цвет \*/

color: white; /\* белый цвет \*/

Ещё один вариант указания цвета — в виде [шестнадцатеричного значения](https://ru.wikipedia.org/wiki/%D0%A8%D0%B5%D1%81%D1%82%D0%BD%D0%B0%D0%B4%D1%86%D0%B0%D1%82%D0%B5%D1%80%D0%B8%D1%87%D0%BD%D0%B0%D1%8F_%D1%81%D0%B8%D1%81%D1%82%D0%B5%D0%BC%D0%B0_%D1%81%D1%87%D0%B8%D1%81%D0%BB%D0%B5%D0%BD%D0%B8%D1%8F). Именно им мы пользовались в [прошлом задании](https://htmlacademy.ru/courses/309/run/10). В этом случае цвет формируется из *красной*, *зелёной* и *синей* составляющих, заданных в виде шестнадцатеричного числа от 00 до ff. Помимо шести, цветовой код может содержать три знака, в этом случае второй символ в цветовых составляющих дублируется первым:

color: #000000; /\* чёрный цвет \*/

color: #f00; /\* красный цвет, то же что #ff0000 \*/

color: #fff; /\* белый цвет, то же что #ffffff \*/

Если не хочется иметь дело с шестнадцатеричными значениями, можно воспользоваться специальной функцией rgb, в которой указывается цвет в более привычном десятичном виде в диапазоне от 0 до 255 также в виде трёх цветовых составляющих, перечисленных через запятую:

color: rgb(0, 0, 0) /\* чёрный, то же что #000000 \*/

color: rgb(255, 0, 0) /\* красный, то же что #ff0000 \*/

color: rgb(255, 255, 255) /\* белый, то же что #ffffff \*/

У функции rgb есть расширенная версия — rgba. В этом случае помимо указания цвета последним значением указывается степень непрозрачности цвета — alpha. Значение может быть от 0 (полностью прозрачный) до 1 (полностью непрозрачный):

color: rgba(0, 0, 0, 0.5) /\* чёрный, непрозрачный на 50% \*/

color: rgba(255, 0, 0, 0.3) /\* красный, непрозрачный на 30% \*/

color: rgba(255, 255, 255, 0.9) /\* белый, непрозрачный на 90% \*/

# Использование отступов для оформления текста

Чтобы текст был более читаемым, необходимо наличие свободного пространства в блоке для этого текста. Вокруг текста должно быть достаточно «воздуха», он не должен «прилипать» к краям, ему не должно быть «тесно».

За отступы в CSS отвечают два свойства: padding задаёт внутренние отступы в блоке, а margin задаёт внешние отступы. Пример, как можно их использовать:

p {

padding: 10px;

margin: 20px 0;

}

В примере выше задан внутренний отступ со всех сторон 10px, а также внешний отступ сверху и снизу 20px и 0px слева и справа. Это [составные](https://htmlacademy.ru/courses/307/run/7) свойства. Подробно padding и margin разбираются в части «[Блочная модель документа](https://htmlacademy.ru/courses/44)».

Посмотрите на блок aside на нашем сайте. Блоку явно не хватает отступов, давайте добавим их!

# Свойство white-space, управление пробелами

Как вы [уже знаете](https://htmlacademy.ru/courses/301/run/7), 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 — схлопывает лишние пробелы и 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: лишние пробелы и переносы строк схлопываются, текст переносится, пробелы в конце строк удаляются.

На самом деле у свойства по [спецификации](https://www.w3.org/TR/css-text-3/#propdef-white-space) может быть ещё два значения:

* break-spaces — поведение идентично pre-wrap, но при схлопывании пробелов остаются нетронутыми *зарезервированные* пробелы, такие как пробелы в конце строки;
* pre-line — как normal, лишние пробелы схлопываются, но строки переносятся на символе переноса строки, на <br> и по мере заполнения строки.

# Стилизация преформатированного текста

В этом задании давайте стилизуем блок <pre> с дочерним <code>. В задачах вам нужно будет только раскомментировать блоки с кодом. Но давайте подробно разберёмся, что именно мы раскомментируем.

Итак, первая задача.

В первом раскомментируемом правиле для pre и code задаются цвет текста color и фоновый цвет background-color. Их мы разбирали в [одном из прошлых заданий](https://htmlacademy.ru/courses/309/run/10).

Дальше идут свойства border и border-radius, мы уже немного пробовали работать с этими свойствами, так что вам они знакомы.

Ещё в первом раскомментированном правиле осталось свойство font-family. Помните, в [задании](https://htmlacademy.ru/courses/309/run/5) про font-family упоминался моноширинный шрифт? Так вот тут мы этим типом шрифта и воспользовались. Название *моноширинный* хоть и звучит непонятно, но по сути своей не сложно. В моноширинном шрифте все символы занимают одинаковое пространство. Такой шрифт использовался раньше в печатных машинках, а сейчас популярен в редакторах кода. Даже сейчас в редакторе кода, в котором вы проходите задания, вы набираете текст именно моноширинным шрифтом.

Ок, идём дальше. Следующими идут отдельные правила для pre и code, в которых задаются внутренние отступы. С ними [вы тоже уже знакомы](https://htmlacademy.ru/courses/309/run/12).

Перейдём ко второй задаче.

Как вы увидите после выполнения первой задачи, к блоку <code> внутри <pre> применились «лишние» свойства. Давайте их переопределим [по каскаду](https://htmlacademy.ru/courses/307/run/12).

# Свойство text-decoration, подчёркивание и другие эффекты

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

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

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

p {

text-decoration: underline; /\* подчёркнутый текст \*/

}

span {

/\* подчёркнутый и зачёркнутый текст \*/

text-decoration: underline line-through;

}

Свойство text-decoration — [составное](https://htmlacademy.ru/courses/307/run/7), а это значит, что его можно разложить на следующие свойства:

* text-decoration-line — вид линии: зачёркивание, подчёркивание или надчёркивание;
* text-decoration-style — стиль линии, может принимать значения:
  + solid — сплошная линия;
  + double — двойная линия;
  + dotted — точечная линия;
  + dashed — пунктирная линия;
  + wavy — волнистая линия.
* text-decoration-color — цвет линии.

**Свойство font-style, курсив**

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

1. normal — обычное начертание;
2. italic — курсивное начертание;
3. oblique — наклонное начертание.

Закономерный вопрос — а чем отличается курсивное и наклонное начертание?

Если задано значение italic, браузер будет пытаться найти в заданном шрифте отдельное курсивное начертание символов. В некоторых шрифтах отдельный курсив предусмотрен.

Если отдельного курсивного начертания в шрифте не предусмотрено, то браузер сделает текст наклонным, то есть сымитирует курсив. Что равноценно заданию тексту значения font-style: oblique.

Закрепим теорию практикой.

**Свойство text-transform, регистр символов**

С помощью CSS можно также управлять регистром символов: делать буквы строчными (маленькими) или заглавными (большими). Делается это с помощью свойства text-transform. Его значения:

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

Без лишних слов — к практике!

# Шлифуем футер

Вот мы и освоили основные CSS-свойства для оформления текста. Наш сайт почти полностью оформлен.

Давайте ещё сделаем последний штрих и добавим стилизацию к подвалу сайта. Для этого мы используем уже известные вам свойства: color, border и padding.

На этом тренажёр подходит к концу, вы закончили разметку и стилизацию целого сайта! Поздравляем! А по [этой ссылке](https://htmlacademy.ru/assets/courses/309/project-state-final.zip?n) можно скачать финальное состояние вёрстки.

# Конспект «Оформление текста»

## Свойство font-size

Управляет размером шрифта. Значение свойства задаёт желаемую высоту символа шрифта. Причём единицы измерения могут быть [абсолютными или относительными](https://htmlacademy.ru/courses/307/run/8).

Самая часто используемая единица измерения размера шрифта — пиксели px:

p {

font-size: 20px;

}

Но, чтобы при изменении основного размера шрифта родителя его дочерние элементы пропорционально меняли свои размеры шрифта, есть специальная единица измерения — em.

Величина 1em — это *такой же* размер шрифта, что и у родителя. Соответственно, если нужно, чтобы шрифт дочернего элемента был всегда в 2 раза больше родительского, то надо задать значение 2em:

h1 {

font-size: 2em;

}

## Свойство line-height

Свойство управляет высотой строки или межстрочным интервалом.

По умолчанию это свойство имеет значение normal. Оно указывает браузеру, что межстрочный интервал нужно подобрать автоматически, исходя из размера шрифта. Спецификация рекомендует устанавливать его в пределах 100-120% от размера шрифта. То есть:

p {

font-size: 10px;

line-height: normal; /\* значение будет примерно 12px \*/

}

Значение normal позволяет всем нестилизованным текстам выглядеть удобочитаемо. Однако, если есть необходимость отойти от стилизации по умолчанию, line-height можно задать фиксированное абсолютное значение в px.

p {

font-size: 16px;

line-height: 26px;

}

Если нужно задать line-height относительное значение, но не такое, как normal, то значение задаётся в процентах или в виде множителя. В таком случае браузер вычисляет значение динамически в зависимости от font-size:

p {

font-size: 10px;

line-height: 150%; /\* вычисленное значение: 10px \* 150% = 15px \*/

line-height: 2; /\* вычисленное значение: 10px \* 2 = 20px \*/

}

Относительные значения более гибкие, чем абсолютные. Но для простых сайтов фиксированных font-size и line-height будет вполне достаточно.

## Свойство font-family

В значении этого свойства указывается список названий шрифтов, перечисленных через запятую. В начале списка располагают самый желаемый шрифт, затем менее желаемый, а в самом конце списка — общий тип шрифта. Браузер проходит по списку слева направо и использует первый найденный в системе или на сайте шрифт. Если название шрифта состоит из нескольких слов, то его нужно заключать в кавычки.

body {

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

}

## Свойство font-weight

Свойство задаёт насыщенность или толщину шрифта. Шрифт может быть жирнее или тоньше обычного начертания. В качестве значения можно использовать ключевое слово или число. Самые часто встречающиеся значения:

* 400 или normal — обычный шрифт, значение по умолчанию;
* 700 или bold — жирный шрифт.

Например:

h1 {

font-weight: 400; /\* то же самое что и normal \*/

}

p {

font-weight: bold; /\* то же самое что и 700 \*/

}

## Свойство text-align

Описывает, как выравнивается текст и другие инлайновые элементы (изображения, инлайн-блоки, инлайн-таблицы и другие) внутри блока по горизонтали.

Свойство может принимать следующие значения:

1. left — выравнивание по левому краю блока, это значение по умолчанию;
2. right — по правому краю блока;
3. center — по центру блока;
4. justify — по ширине блока, при этом слова в строке будут размещаться так, чтобы занять равномерно всё пространство строки (пробелы между словами в таком случае становятся неравномерными, так как браузер «растягивает» слова в строке).

Важно помнить, что свойство text-align применяется именно к самому блоку-контейнеру, внутри которого находится текстовый контент:

HTML:

<p>

Я текст внутри абзаца

</p>

CSS:

p {

text-align: center;

}

## Свойство vertical-align

Свойством можно выравнивать *инлайновые* элементы относительно содержащей его строки. Самый простой пример — выровнять картинку <img> по вертикали в текстовой строке.

У свойства vertical-align много значений, но самые часто используемые:

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

В отличие от text-align свойство vertical-align задаётся самому элементу, а не содержащему его контейнеру:

HTML:

<p>

<img src="picture.png" alt="Я картинка">

Я текст внутри абзаца

</p>

CSS:

img {

vertical-align: middle;

}

## Свойство color

Цветом текста можно управлять свойством color.

Цвет может быть задан в виде ключевого слова (полный список ключевых слов приводится в [спецификации](https://www.w3.org/TR/css-color-3/#svg-color)). Например:

color: black; /\* чёрный цвет \*/

color: red; /\* красный цвет \*/

color: white; /\* белый цвет \*/

Ещё один вариант указания цвета — в виде [шестнадцатеричного значения](https://ru.wikipedia.org/wiki/%D0%A8%D0%B5%D1%81%D1%82%D0%BD%D0%B0%D0%B4%D1%86%D0%B0%D1%82%D0%B5%D1%80%D0%B8%D1%87%D0%BD%D0%B0%D1%8F_%D1%81%D0%B8%D1%81%D1%82%D0%B5%D0%BC%D0%B0_%D1%81%D1%87%D0%B8%D1%81%D0%BB%D0%B5%D0%BD%D0%B8%D1%8F). В этом случае цвет формируется из *красной*, *зелёной* и *синей* составляющих, заданных в виде шестнадцатеричного числа от 00 до ff. Помимо шести, цветовой код может содержать три знака, в этом случае второй символ в цветовых составляющих дублируется первым:

color: #000000; /\* чёрный цвет \*/

color: #f00; /\* красный цвет, то же что #ff0000 \*/

color: #fff; /\* белый цвет, то же что #ffffff \*/

Если не хочется иметь дело с шестнадцатеричными значениями, можно воспользоваться специальной функцией rgb, в которой указывается цвет в более привычном десятичном виде в диапазоне от 0 до 255 также в виде трёх цветовых составляющих, перечисленных через запятую:

color: rgb(0, 0, 0) /\* чёрный, то же что #000000 \*/

color: rgb(255, 0, 0) /\* красный, то же что #ff0000 \*/

color: rgb(255, 255, 255) /\* белый, то же что #ffffff \*/

У функции rgb есть расширенная версия — rgba. В этом случае помимо указания цвета последним значением указывается степень непрозрачности цвета — alpha. Значение может быть от 0 (полностью прозрачный) до 1 (полностью непрозрачный):

color: rgba(0, 0, 0, 0.5) /\* чёрный, непрозрачный на 50% \*/

color: rgba(255, 0, 0, 0.3) /\* красный, непрозрачный на 30% \*/

color: rgba(255, 255, 255, 0.9) /\* белый, непрозрачный на 90% \*/

## Контраст цвета текста и фона

Фоновое изображение и фоновый цвет блока всегда должен достаточно сильно контрастировать с цветом текста в этом блоке. Чем больше контраст, тем удобнее этот текст читать в разных условиях освещённости и на разных устройствах. Поэтому если вы задаёте блоку фоновое изображение, нужно обязательно дополнительно задавать подходящий фоновый цвет. В этом случае, пока изображение загружается, или в случае, если оно совсем не загрузится, текст всё равно можно будет прочитать:

p {

/\* идеальный контраст: цвет текста белый, цвет фона — чёрный \*/

background-color: #000000;

color: #ffffff;

}

span {

/\* плохой контраст: цвет текста и фона — серые \*/

background-color: #cccccc;

color: #dddddd;

}

## Свойство white-space, управление пробелами

Браузер игнорирует множественные пробелы и переносы строк в HTML-коде. С помощью свойства white-space можно управлять пробелами и переносами строк. Свойство принимает значения:

* nowrap — схлопывает лишние пробелы и 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: лишние пробелы и переносы строк схлопываются, текст переносится, пробелы в конце строк удаляются.

## Свойство text-decoration

Задаёт дополнительное оформление текста. Значения свойства:

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

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

p {

text-decoration: underline; /\* подчёркнутый текст \*/

}

span {

/\* подчёркнутый и зачёркнутый текст \*/

text-decoration: underline line-through;

}

Свойство text-decoration — [составное](https://htmlacademy.ru/courses/307/run/7). Оно раскладывается на отдельные свойства:

* text-decoration-line — вид линии: зачёркивание, подчёркивание или надчёркивание;
* text-decoration-style — стиль линии, может принимать значения:
  + solid — сплошная линия;
  + double — двойная линия;
  + dotted — точечная линия;
  + dashed — пунктирная линия;
  + wavy — волнистая линия.
* text-decoration-color — цвет линии.

## Свойство font-style

Свойством можно задать начертание текста. Его основные значения:

1. normal — обычное начертание;
2. italic — курсивное начертание;
3. oblique — наклонное начертание.

Если задано значение italic, браузер будет пытаться найти в заданном шрифте отдельное курсивное начертание символов. В некоторых шрифтах отдельный курсив предусмотрен.

Если отдельного курсивного начертания в шрифте не предусмотрено, то браузер сделает текст наклонным, то есть сымитирует курсив. Что равноценно заданию тексту значения font-style: oblique.

## Свойство text-transform

С его помощью можно управлять регистром символов: делать буквы строчными (маленькими) или заглавными (большими). Значения свойства:

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

## Отступы

Важный фактор того, что текст в блоке будет удобочитаемым, это наличие свободного пространства в блоке для этого текста. Вокруг текста должно быть достаточно «воздуха», он не должен «прилипать» к краям, ему не должно быть «тесно».

За отступы в CSS отвечают два свойства: padding задаёт внутренние отступы в блоке, а margin задаёт внешние отступы.

!!!

# Конспект «Основы HTML и CSS»

## HTML

HTML расшифровывается как «Hypertext Markup Language», то есть «язык гипертекстовой разметки».

Язык HTML отвечает за структуру и содержание страницы. HTML состоит из тегов, а тег состоит из имени, заключённого между знаками «меньше» и «больше». Примеры тегов: <h1>, <p>, <ul>.

### **Парные теги**

Теги бывают парными и одиночными. Парные теги могут содержать текст и другие теги. У парных тегов, в отличие от одиночных, есть вторая половинка — закрывающий тег:

<h1>Текст заголовка</h1>

В закрывающей части парных тегов перед именем ставится символ / («слэш»).

В парные теги можно вкладывать другие теги. Например, как в списках:

<ul>

<li>Элемент списка</li>

</ul>

У вложенных тегов всегда нужно следить за правильным порядком закрытия. Вложенный тег не может закрываться позже родительского:

<ul><li>Элемент списка</ul></li> <!-- Плохо -->

<ul><li>Элемент списка</li></ul> <!-- Хорошо -->

Не все теги можно вкладывать в другие теги, например, тег <h1> нельзя вкладывать в <p>.

### **Одиночные теги**

Существуют не только парные, но и одиночные теги. Например, тег <img> позволяет добавить картинку в разметку.

Сам по себе <img> не имеет смысла. Чтобы этот тег был действительно полезен, необходимо написать внутри него адрес, ведущий к картинке. Делается это с помощью атрибута src:

<img src="keks.png">

У тега может быть несколько атрибутов. В этом случае они пишутся через пробел:

<тег атрибут1="значение1" атрибут2="значение2">

Например, картинке при желании можно задать размеры:

<img src="keks.png" width="200" height="100">

### **Комментарии**

Код, заключённый между символами <!-- и -->, работать не будет. Если эти символы удалить, то код заработает, это называется «раскомментировать». С помощью комментариев обычно временно отключают какой-то код или оставляют подсказки и разъяснения.

<!-- Это комментарий в HTML -->

## CSS

CSS расшифровывается как «Cascading Style Sheets», то есть «каскадные таблицы стилей».

Язык CSS отвечает за внешний вид страницы.

С помощью CSS можно задавать параметры для любого тега: ширину и высоту, отступы, цвет и размер шрифта, фон и так далее. Все эти параметры задаются с помощью свойств в следующем формате:

свойство: значение;

Например:

color: red;

padding: 10px;

Стили к тегам добавляются чаще всего при помощи атрибута class.

Например, если мы хотим, чтобы определённые стили, описанные, допустим, в классе feature-kitten, применились к тегу <p>, то в разметке напишем так:

<p class="feature-kitten">...</p>

### **CSS-правила**

Можно сказать, что CSS-правило — это группа свойств и их значений, которая целиком применяется к тем тегам, на которые указывает селектор.

И выглядит это так:

селектор {

свойство1: значение;

свойство2: значение;

}

Задавать стили можно не только с помощью атрибута class, но и по тегам. Селектор указывает, к каким тегам применятся свойства из CSS-правила. Селекторы по тегам работают проще всего: они выбирают все теги с подходящим именем.

p { color: red; }

В примере селектором является p, и он выбирает все теги с именем p (то есть теги <p>), а теги с другим именем, например h1, не выбирает.

Когда же стилизация задаётся по классам, то стили применяются только к тегам с такими классами.

.название\_класса {

свойство: значение;

}

### **Миксование классов**

У HTML-элемента может быть сколько угодно классов, в этом случае они перечисляются в атрибуте class через пробел, например:

<li class="product">Товар</li>

<li class="product hit">Товар, а ещё хит продаж</li>

<li class="product hit sale">Товар, хит продаж и со ски-и-идкой!</li>

Обычно миксование используют так: в один класс выносят общее оформление, а в дополнительных классах описывают его модификации.

### **Комментарии**

В CSS тоже существуют комментарии, их отличие от HTML-комментариев в том, что код или подсказки пишутся между символами /\* и \*/.