**5.2. Подключение стилей к HTML**

Абсолютное большинство современных сайтов использует **CSS** (каскадные таблицы стилей). Этот язык отвечает за визуальное оформление элементов, помогает настроить шрифты и цвета, расположить блоки.

**Получается, что HTML отвечает за разделение страницы, а CSS — за визуальное представление.**

Но для того чтобы таблицы стилей начали работать, их нужно подключить к разметке. Это можно сделать тремя способами:

* 1. Использовать **глобальные** стили *CSS*;
  2. Использовать **внутренний** *CSS* для конкретного элемента страницы;
  3. Подключать **внешние** файлы с расширением .css.

**Глобальные стили**

Свойства *CSS* при использовании **глобальных**стилей описывают внутри тега <style>, который, в свою очередь, располагается внутри тега <head>. Стили будут активны только на одной конкретной странице.

Выглядит это так:

<!DOCTYPE HTML>

<html>

<head>

<meta charset="utf-8">

<style>

h1 {

**font-size**: **22px**;

**color**: red;

}

body {

**background-color**: blue;

}

</style>

</head>

<body>

<h1>Hello, world!</h1>

</body>

</html>

Стили, подключённые таким образом, загружаются заново при каждой перезагрузке страницы. Это может повлиять на скорость взаимодействия пользователя с сервисом, поэтому в реальных и больших проектах **к глобальным стилям прибегают очень редко**.

Впрочем, этот способ можно использовать, если общих стилей на странице немного. Например, так удобно задавать цвет фона, как показано в нашем примере.

*Для демонстрации работы кода мы используем сервис JSFiddle. В основном окошке вы видите код, во вкладке****Result****— результат его исполнения. Более того, вы можете попробовать дополнить или изменить код самостоятельно. Для этого нажмите в правом верхнем углу виджета кнопку****Edit in JSFiddle****. В открывшемся редакторе вы можете работать. Чтобы посмотреть, как ваши изменения воздействуют на результат, не забудьте нажать на****Run****(в левой части шапки).*

**Внутренние стили**

**Внутренние**стили используются для конкретного тега HTML, реализуются при помощи атрибута style.

<!DOCTYPE HTML>

<html>

<head>

...

</head>

<body>

<h1 style="font-size: 22px; color: red">Hello, world!</h1>

</body>

</html>

Этот способ, который иначе называется**инлайн-стилями**, удобен для проверки и предпросмотра изменений. Однако управление сайтом может стать довольно трудным: изменения придётся вносить отдельно для каждого элемента.

**Внешние стили**

Чаще всего **внешние** стили подключают из CSS-файла с помощью тега <link>. Все изменения в исходном файле будут отражаться на сайте.

<!DOCTYPE HTML>

<html>

<head>

<link href="style.css" rel="stylesheet" type="text/css">

<link href="http://ourwebsite.ru/main.css" rel="stylesheet">

</head>

<body>

<h1>Hello, world!</h1>

</body>

</html>

В атрибуте href задаётся путь к *CSS*-файлу.

Атрибут rel (*relation*) указывает тип связи между документом и файлом.

rel="stylesheet" сообщает браузеру, что мы подключаем стили.

type="text/css" указывает тип внешнего файла.

Тег <link> будет работать и в другом месте страницы, но обычно стили подключают внутри <head>. Так проще увидеть их все одновременно.

Для разных страниц можно использовать один и тот же *CSS*-файл, который может выглядеть так:

h1 {

**font-size**: **22px**;

**color**: red;

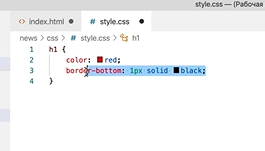
}

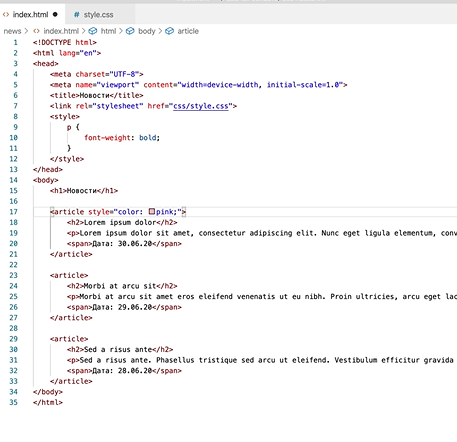
body {

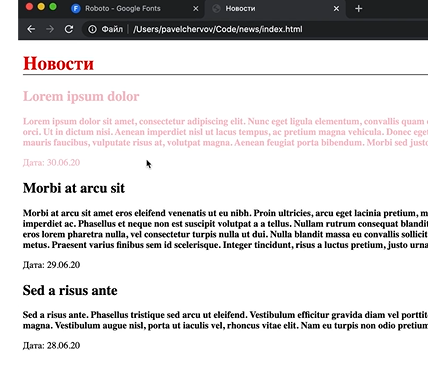
**background-color**: blue;

}

В этом видео мы на практике разберёмся, как создавать внешние, внутренние и встроенные таблицы стилей *CSS*.







**5.3. Синтаксис CSS. Простые и комбинированные селекторы**

**Синтаксис CSS**

Правила стилей записываются в особом формате, который называется *CSS*. При этом стили почти всегда связаны с *HTML* благодаря [селекторам](https://developer.mozilla.org/ru/docs/Web/CSS/CSS_%D0%A1%D0%B5%D0%BB%D0%B5%D0%BA%D1%82%D0%BE%D1%80%D1%8B).

Что такое селектор?

**Селектор**определяет, какой элемент (или элементы) станет объектом определения стилей. **Свойство**селектораотвечает за выбор характеристики для изменений. И наконец, его **значение**помогает определить, как именно что-либо необходимо изменить. Вся эта запись называется **правило CSS**.

**Запись правила CSS** в общем виде выглядит так:

селектор {

свойство-1: значение-1;

свойство-2: значение-2;

}

Используя конкретные параметры, можно записать следующим образом:

h1 {

**color**: red;

**background**: blue;

}

То есть самый крупный заголовок на странице будет окрашен красным, а его фон станет синим.

Обратите внимание, что фигурные скобки и точка с запятой после значения (или после последнего значения, если свойство это подразумевает) **обязательны**!

**Виды селекторов**

Список **свойств** и **значений** *CSS* достаточно длинный, но при этом он заранее определён. Нужно просто познакомиться с большинством этих параметров, используя, например, [документацию](https://developer.mozilla.org/en-US/docs/Web/CSS/Reference#Keyword_index).

С **селекторами,**то есть способами связи элементов *HTML* и назначенных им стилей,ситуация иная. Вопрос выбора, систематизация этого процесса и последствия целиком на совести разработчика.

При этом, с одной стороны, **нужно избегать использования слишком общих селекторов**. Например, если назначить какие-то параметры для <p> (параграфа), то они будут применены ко всем параграфам на странице без исключений. С другой стороны, **необходимо не слишком вдаваться в частности**, иначе для внесения небольших изменений придётся проделывать большой объём работы.

Селекторы можно разделить на типы:

* 1. Простые,
  2. Комбинированные,
  3. Селекторы псевдоклассов,
  4. Селекторы псевдоэлементов.

Рассмотри первые два.

**Простые селекторы**

В свою очередь, простые селекторы делятся на:

* + Селектор по тегу,
  + Селектор по идентификатору,
  + Селектор по атрибуту,
  + Селектор по классу.

Самый используемый вариант — последний.

Кроме того, к этой группе относится **универсальный** селектор, который обозначается звёздочкой \* и предназначен для работы со всеми элементами страницы.

1

**Селектор по тегу** позволяет выбрать на странице все элементы какого-либо типа, например, параграфы, заголовки, блоки и так далее. Для использования этого типа простых селекторов надо записать тег без скобок, как вы уже видели в примере с покраской шрифта заголовка.

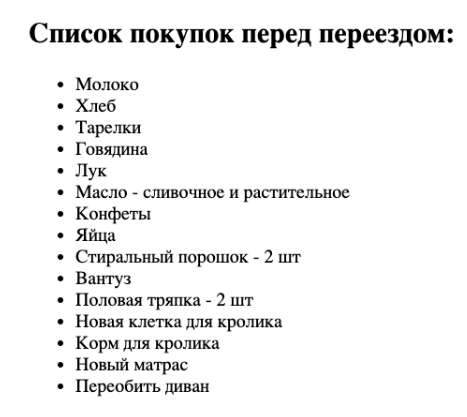
h1 {

**color**: red;

}

2

**Селектор по классу**— самый распространённый вариант в современной вёрстке. Он позволяет разработчику самостоятельно выбрать элементы, к которым будет применён стиль. Правда, сначала придётся внести изменения в *HTML*. Представим, что вы готовитесь к переезду, и для того, чтобы подсчитать бюджет, накидали общий список покупок.



Теперь же настала пора отправляться за покупками, и нужно как-то выделить то, что вы собираетесь приобрести в продовольственном, хозяйственном и магазине для животных. Для этого в*HTML* после каждого тега li до закрывающей скобки нужно написать class="food", "household" или "vet".

**К названиям классов есть следующие требования:**

* Класс не может начинаться с цифры.
* Класс не может содержать пробел.
* Класс нужно называть буквами латинского алфавита.
* Класс чувствителен к регистру: Food и food — два разных класса.
* Класс может начинаться и содержать дефис ‐ и/или нижнее подчеркивание \_.

После дополнения классами внешний вид списка никак не изменится. Но стоит применить *CSS*, как страница заиграет новыми красками. Допустим, все пункты списка, относящиеся к еде, станут оранжевыми, все хозяйственные принадлежности — зелёными, предметы первой необходимости для кролика — голубыми. Для этого мы применим селекторы: .food, .household и .vet. **Точка перед названием — специальное обозначение, которое даёт CSS понять, что мы применяем стиль к классу.**

|  |  |  |
| --- | --- | --- |
| <h2>Список покупок перед переездом:</h2>  <ul>  <li class="food">Молоко</li>  <li class="food">Хлеб</li>  <li class="household">Тарелки</li>  <li class ="food">Говядина</li>  <li class ="food">Лук</li>  <li class ="food">Масло - сливочное и растительное</li>  <li class ="food">Конфеты</li>  <li class ="food">Яйца</li>  <li class ="household">Стиральный порошок - 2 шт</li>  <li class ="household">Вантуз</li>  <li class ="household">Половая тряпка - 2 шт</li>  <li class="vet">Новая клетка для кролика</li>  <li class="vet">Корм для кролика</li>  <li class="household">Новый матрас</li>  <li>Переобить диван</li>  </ul> | **.food** {  **color**: orange;  }  **.household** {  **color**: green;  }  **.vet** {  **color**: lightblue;  }  **.not-bought** {  **background**: red;  **color**: white;  } |  |

После похода в ближайшие магазины выяснилось, что некоторые пункты так просто не добыть. Для того чтобы их отметить, можно ввести дополнительный класс. Например, not-bought. Добавлять его в *HTML* нужно к существующим классам через пробел в те же кавычки. После этого можно отметить недостающие покупки, допустим, красным цветом фона.

Обратите внимание: селекторы можно записывать последовательно, без пробела. Например:

**.class1.class2** /\* элемент с двумя классами class1 и class2 \*/

div**#some-id.class1** /\* div с идентификатором some-id и классом class1\*/

Из всего списка покупок неохваченной классами осталась только переобивка дивана. Она предполагает не поход в магазин, а обращение в сервисную службу. Порядок действий в данном случае будет несколько отличаться. Для таких особенных элементов существует понятие идентификатор.

|  |  |
| --- | --- |
| <nav>  <a href="#Food">Продукты</a>  <a href="#Household">Хозтовары</a>  <a href="#Vet">Для кролика</a>  <a href="#Service">Сервис</a>  </nav>  <h2>Список покупок перед переездом:</h2>  <ul id="Food"><strong>Продукты</strong>  <li>Молоко</li>  <li>Хлеб</li>  <li>Говядина</li>  <li>Лук</li>  <li>Масло - сливочное и растительное</li>  <li>Конфеты</li>  <li>Яйца</li></ul>  <ul id="Household"><strong>Хозтовары</strong>  <li>Тарелки</li>  <li>Стиральный порошок - 2 шт</li>  <li>Вантуз</li>  <li>Половая тряпка - 2 шт</li>  <li>Новый матрас</li>  </ul>  <ul id="Vet"><strong>Для кролика</strong>  <li>Новая клетка для кролика</li>  <li>Корм для кролика</li></ul>  <ul id="Service"><strong>Сервис</strong>  <li>Переобить диван</li>  </ul> |  |

3

**Селектор по идентификатору** определяет уникальное имя элемента, которое используется для изменения его стиля и обращения к нему через скрипты. В *HTML* он обозначается при помощи букв id:

**Идентификатор**

<li id="Sofa">Переобивка дивана</li>

В *CSS* для записи селектора по идентификатору используется решётка.

**#Sofa** {

**color**: red;

}

Имя идентификатора выбирают, следуя тем же правилам, что и для классов:

* Не может начинаться с цифры.
* Не может содержать пробел.
* Нужно называть буквами латинского алфавита.
* Чувствителен к регистру: Sofa и sofa — два разных идентификатора.
* Может начинаться и содержать дефис - и/или нижнее подчеркивание \_.

Учитывая, что основное свойство идентификатора — уникальность, его применение для целей назначения стилей должно быть очень ограниченным. Излишняя специфичность приведёт к большей трудоёмкости (только представьте, сколько стилей придётся назначать идентификаторам в большом проекте). Кроме того, именно у идентификатора самый большой приоритет среди селекторов, то есть стили, назначенные при помощи селектора по идентификатору, очень сложно менять.

Основное применение идентификатора вы уже рассматривали в предыдущем модуле. Речь про навигацию по странице и создание якорей. Список покупок при помощи этого метода можно переформатировать следующим образом.

После появления в списке навигации стало очень легко стилизовать отдельные маленькие списки. А вот как быть с самим меню? Можно использовать селектор по тегу а, но мы подразумеваем, что проекты, с которыми вам предстоит работать, будут больше списка покупок, так что ссылок в них окажется не четыре. Можно каждой ссылке присвоить класс, но в данном случае есть прямой путь.

4

**Селектор по атрибуту** позволяет обратиться к элементу *HTML* по значению его атрибутов, записывается при помощи квадратных скобок.

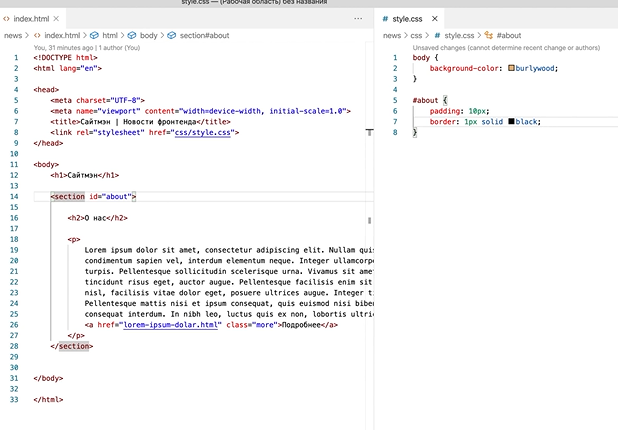
В нашем примере мы будем обращаться к ссылкам, используя атрибут href и его значение.  Например, для того, чтобы изменить цвет фона ссылки на продукты, используем селектор a[href="#Food"].

Можно обратиться по атрибуту, зная только часть значения. Допустим, вы забыли, как по-английски хозтовары. *House*... что-то там, да ещё и не помню, с какой буквы. Тогда селектор можно записать следующим образом a[href \*= "ouse"] или a[href \*= ouse], а фон сделаем зелёным. Оператор \*= означает, что искомое может располагаться где угодно в значении атрибута.

Есть отдельные операторы, которые позволяют обратиться к атрибуту, зная только окончание значения $=, или только его начало ^=.  С другими вариантами селекторов по атрибуту вы можете познакомиться в [документации](https://developer.mozilla.org/en-US/docs/Web/CSS/Attribute_selectors).

В реальной жизни селекторы по атрибуту нужны не столько для борьбы с забывчивостью, сколько для **регулирования поведения элементов**, внешний вид которых различается в зависимости от атрибутов. Например, речь может идти о формах, с которыми вы познакомитесь позже.

|  |  |  |
| --- | --- | --- |
| <nav>  <a href="#Food">Продукты</a>  <a href="#Household">Хозтовары</a>  <a href="#Vet">Для кролика</a>  <a href="#Service">Сервис</a>  </nav>  <h2>Список покупок перед переездом:</h2>  <ul id="Food"><strong>Продукты</strong>  <li>Молоко</li>  <li>Хлеб</li>  <li>Говядина</li>  <li>Лук</li>  <li>Масло - сливочное и растительное</li>  <li>Конфеты</li>  <li>Яйца</li></ul>  <ul id="Household"><strong>Хозтовары</strong>  <li>Тарелки</li>  <li>Стиральный порошок - 2 шт</li>  <li>Вантуз</li>  <li>Половая тряпка - 2 шт</li>  <li>Новый матрас</li>  </ul>  <ul id="Vet"><strong>Для кролика</strong>  <li>Новая клетка для кролика</li>  <li>Корм для кролика</li></ul>  <ul id="Service"><strong>Сервис</strong>  <li>Переобить диван</li>  </ul> | a[href="#Food"] {  **background**: orange;  }  a[href\*="ouse"] {  **background**: green;  } |  |



## ****Комбинированные селекторы****

Комбинированные селекторы образуются из простых селекторов по специальным правилам.

**Простой комбинатор (запятая)**выбирает все совпадающие узлы.

Синтаксис: селектор-1, селектор-2

Пример: div, p выберет оба элемента: и <div>, и <p>.

**Комбинатор потомков** **(пробел)** выбирает элементы, которые находятся внутри указанного элемента (вне зависимости от уровня вложенности).

Синтаксис: селектор-1 селектор-2 (через пробел)

Пример: div p выберет все элементы <p>, которые находятся внутри элемента <div>.

**Комбинатор дочерних элементов (>)**, в отличие от пробела, выбирает только дочерние элементы.

Синтаксис: селектор-1 > селектор-2

Пример: div > img выберет только дочерние, первого уровня вложенности, элементы <img>.

**Комбинатор соседних элементов (~)** выбирает элементы, которые находятся на этом же уровне вложенности, после указанного элемента, с тем же родителем.

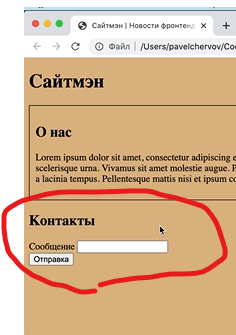
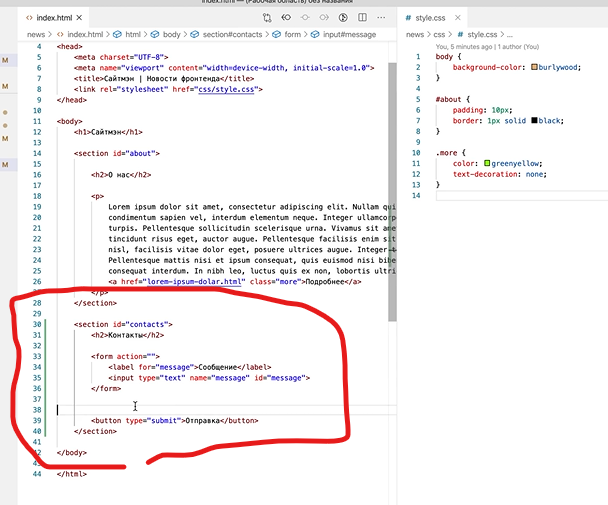
Синтаксис: селектор-1 ~ селектор-2

Пример: p ~ .cat выберет все элементы с классом cat, которые находятся после элемента <p> внутри одного родителя.

**Комбинатор следующего соседнего элемента (+)** выбирает элемент, который находится сразу после указанного элемента, внутри одного родителя.

Синтаксис: селектор-1 + селектор-2

Пример: .header + .container выберет элемент с классом container, который находится непосредственно после элемента с классом header.



## ****Комбинированные селекторы****

## 5.4. Селекторы псевдоклассов и псевдоэлементов

Разберём оставшиеся типы селекторов. Начнём с селекторов псевдоклассов.

**Псевдоклассы**определяют специальные состояния элементов. Они обязательно относятся к какому-либо селектору, записываются через двоеточие.

 Например:

a**:visited** {

**color**: red;

}

Полный список псевдоклассов можно найти [здесь](https://developer.mozilla.org/en-US/docs/Web/CSS/Pseudo-classes).

А мы поговорим про самые частые:

* + **:hover** (наведение на объект, чаще всего кнопку или ссылку).
  + **:visited** (посещённая ранее ссылка).
  + **:focus** (элемент с фокусом на нём, например, элемент формы).
  + **:first-child** и **:last-child** (первый и последний подобный встречающийся элемент, например, li:first-child первый элемент списка).

**:nth-child**(число) (касается конкретного элемента, например, li:nth-child(2) — второй элемент <li>).

:**nth-child(odd)** (все нечётные элементы).

**:nth-child(even)** (все чётные элементы).

* + **:first-of-type** и **:nth-of-type**(первый и *n*-ный элемент какого-либо типа).
  + **:disabled** (недоступный элемент, например, button:disabled — кнопка, на которую невозможно нажать).

В примере при наведении на пункты меню фон становится зелёным, при наведении на списки — оранжевым. Так работает псевдокласс **hover**.

Пункты списков окрашены при помощи псевдокласса **:nth-child (3n+1).** Обратите внимание, что **в разработке значения начинаются с нуля**, то есть счётчик *n* исчисляется как 0, 1, 2, 3..., поэтому первые пункты тоже оказались покрашены.

**Псевдоэлементы**

Наравне с псевдоклассами, **псевдоэлементы** используются для того, чтобы сделать страницу более удобной для пользователя. Они стилизуют часть выбранного элемента. Записываются после двух двоеточий без пробела.

**::first-line** может изменить стиль первой строки. Например, абзаца.

**::before** добавляет что-то перед началом элемента. Например, в примере со списком покупок, если бы появились какие-то новые пункты, мы могли бы создать для них специальный класс (в *HTML*: class="newads"), а потом поставить для обозначения какой-то символ, используя  .newads::before.

**::after** добавляет что-то после окончания элемента.

Двойное двоеточие :: для псевдоэлементов появилось в *CSS3*, чтобы отличать от псевдоклассов. Раньше же использовалось одинарное. Тем не менее для совместимости со старым синтаксисом большинство браузеров допускает оба варианта :: и :. Таким образом, вы можете использовать любой из способов написания.

Полный список псевдоэлементов можно найти [здесь](https://developer.mozilla.org/ru/docs/Web/CSS/Pseudo-elements#%D0%A1%D0%BF%D0%B8%D1%81%D0%BE%D0%BA_%D1%81%D1%82%D0%B0%D0%BD%D0%B4%D0%B0%D1%80%D1%82%D0%BD%D1%8B%D1%85_%D0%BF%D1%81%D0%B5%D0%B2%D0%B4%D0%BE%D1%8D%D0%BB%D0%B5%D0%BC%D0%B5%D0%BD%D1%82%D0%BE%D0%B2).

## 5.5. Приоритет селекторов

Красный цвет шрифта ссылки мы хотели задать по **тегу** a { color: red; }, зелёным её должен был сделать **селектор класса** .main { color: green; }. Но в итоге ссылка стала синей из-за **селектора по идентификатору** #link { color: blue; }.

Почему?

Дело в [специфичности селекторов](https://developer.mozilla.org/ru/docs/Web/CSS/Specificity), которая определяет, какой селектор будет приоритетнее и какие именно стили будут применены к элементу.

Помимо специфичности, эту особенность называют **приоритетом** или **весом** селектора (наиболее используемый термин), который придаёт стилевые свойства *HTML-*тегу.

* + Селектор тега весит 1 балл.
  + Селектор класса, селектор по атрибуту или селектор псевдокласса — 10 баллов.
  + Селектор по идентификатору — 100 баллов.
  + Внутренний стиль (стиль, заданный при помощи атрибута style в коде *HTML*) — 1000 баллов.

Чем больше вес, тем выше вероятность, что стиль, который собирается назначить селектор, «победит» остальные.

Источник: [css.yoksel.ru](http://css.yoksel.ru/specifity/" \t "_blank)

Допустим, есть меню, попробуем посчитать вес селекторов на этом примере.

<ul class="menu">

<li class="menu-item">

first

<li>

<li class="menu-item">

second

<li>

<ul>

При обращении в *CSS* к элементу списка, используя селектор по тегу li, мы получим следующий вес:

li — 0,0,0,1.

Обратившись аналогично к этому же элементу, но используя селектор по тегу родителя, мы получим:

ul li — 0,0,0,2.

И этот стиль перевесит предыдущий, так как селектор с двумя тегами весит больше, чем с одним.

То есть этот стиль для font-size:

li {

**font-size**: **14px**;

}

Будет перезаписан вот этим:

ul li {

**font-size**: **16px**;

}

Если дописать стиль с селектором по классу, который  весит больше, чем селектор с тегом, то есть вес =.item  0,0,1,0.

font-size будет перезаписан, и размер в итоге будет равен 18*px*.

**.item** {

**font-size**: **18px**;

}

**Обратите внимание**

* 1. Специфичность имеет значение только в том случае, если один элемент подходит сразу нескольким селекторам.
  2. Если у нескольких правил специфичность одинакова, то к элементу применяется последнее по порядку правило *CSS*.
  3. Правило для конкретного элемента всегда будет иметь больший приоритет, чем правила, унаследованные от его предка.

Чтобы узнать, какие селекторы имеют большую специфичность, можно воспользоваться одним из калькуляторов, таких как [этот](https://www.codecaptain.io/tools/css-specificity-calculator) или [этот](https://specificity.keegan.st/).

**Повысить специфичность селекторов**можно, указав один или несколько элементов, предшествующих необходимому в дереве документа. Или, например, указав в одном селекторе сразу несколько классов, которым принадлежит нужный элемент.

Цвет шрифта ссылок остался чёрным, а цвет фона — зелёным.

**Модификатор !important**

Запрещённый, но при этом вполне работающий приём при определении стилей — применение **модификатора !important**.

Этот модификатор позволяет «перебить» стили, несмотря на вес селекторов.

То есть, если вернуться к примеру выше и для списка определить три блока *CSS:*

li {

**font-size**: **14px**;

}

ul li {

**font-size**: **16px**;

}

**.item** {

**font-size**: **18px**;

}

И в стиле с наименьшим весом селекторов определить правило !important:

li {

**font-size**: **14px** !important;

}

Размер шрифта в таком случае будет равен **14px**.

Если каждому правилу объявить !important, применён будет тот, который будет прочитан браузером последним.

li {

**font-size**: **14px** !important;

}

**.item** {

**font-size**: **18px** !important;

}

ul li {

**font-size**: **16px** !important;

}

То есть размер шрифта будет равен  **16px**.

Cуть работы !important в том, что стиль получит наивысший приоритет, но будет нарушено естественное каскадирование, что затруднит отладку и понимание кода разработчиком.

**Советы разработчика**

Слишком много правил? Именно поэтому в большинстве случаев легче следовать best practices:

* 1. **Используйте преимущественно селекторы классов.** Даже если речь идёт о заголовке или другом уникальном элементе.
  2. **Не применяйте к одному элементу чрезмерное количество классов.**
  3. **Избегайте встроенных стилей**, таких как <a style="color: red;">.
  4. **Не рассчитывайте на модификатор !important.** Лучше, если вы просто будете знать о его существовании, но не более.

### Задание 5.5.1

0/4 points (graded)

Представлен следующий фрагмент HTML-разметки:

<body><a class="home"id="link"href="https://skillfactory.ru/">

Skill Factory </a>

<img class="home" id=”picture” src="img.jpg">

</body>

Напишите:

* 1. Селектор по тегу для посещённой ссылки.  неверно
  2. Селектор по идентификатору для картинки.  неверно
  3. Селектор для всех элементов, с классом home (при помощи простого селектора).  неверно
  4. Cелектор по тегу для всех изображений.  неверно

### Задание 5.5.2

2 points possible (graded)

<aclass="home-link home-tabs\_\_search"data-id="video"href="/profile"target="\_blank"rel="noopener"data-statlog="tabs.video">Video</a>

Определите, какой селектор будет иметь наибольшую специфичность:

a

a.home-link

.home-link.home-tabs\_\_search верно

.home-link

.home-tabs\_\_search

Какие селекторы относятся к данному элементу?

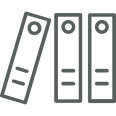
a верно

a.home-link верно

a .home-link

a, .home-link верно

#video

**Дополнительно**

Вы усвоили CSS-классы и селекторы? [Эта игра](http://flukeout.github.io/) поможет вам проверить свои силы!

## 5.6. Цвета в CSS

**Цвета в CSS** могут задаваться с помощью одного из готовых значений-слов: red, yellow, black, blue и так далее. Хотя названия более или менее описывают цвета, строгого обоснования используемых имён нет.

Полный перечень можно посмотреть [здесь](https://www.w3schools.com/colors/colors_names.asp).

HTML распознаёт только 16 основных цветовых ключевых слов из CSS1, а остальные браузер пытается интерпретировать, используя определённый алгоритм. Из-за этого в разных браузерах одно и то же слово может выдавать разные значения.

Стоит также иметь в виду, что, в отличие от HTML, CSS полностью игнорирует неизвестные ключевые слова. Кроме того, количество готовых цветов ограничено.

Все цвета в стилях, даже те, для которых есть названия, представляют цвет в **цветовом спектре sRGB**. Цвет также может включать значение прозрачности [Альфа-канала](https://ru.wikipedia.org/wiki/%D0%90%D0%BB%D1%8C%D1%84%D0%B0-%D0%BA%D0%B0%D0%BD%D0%B0%D0%BB" \l ":~:text=%D0%90%D0%BB%D1%8C%D1%84%D0%B0%2D%D0%BA%D0%B0%D0%BD%D0%B0%D0%BB%2C%20%D1%82%D0%B0%D0%BA%D0%B6%D0%B5%20%D0%B8%D0%B7%D0%B2%D0%B5%D1%81%D1%82%D0%BD%D1%8B%D0%B9%20%D0%BA%D0%B0%D0%BA,%D0%BF%D1%80%D0%BE%D0%B7%D1%80%D0%B0%D1%87%D0%BD%D1%8B%D0%BC%2C%20%D0%BB%D0%B8%D0%B1%D0%BE%20%D0%B0%D0%B1%D1%81%D0%BE%D0%BB%D1%8E%D1%82%D0%BD%D0%BE%20%D0%BD%D0%B5%D0%BF%D1%80%D0%BE%D0%B7%D1%80%D0%B0%D1%87%D0%BD%D1%8B%D0%BC)." \t "_blank), но ключевые слова цвета представляют собой простые цвета без прозрачности.

## ****Цвета в шестнадцатеричном представлении****

**Классический способ задавать** RGB-цвета, который используется не только в веб-разработке, но и во многих графических редакторах, **шестнадцатеричное представление**. На каждый из основных цветов и альфа-канал в таком представлении выделяется по два символа в шестнадцатеричном алфавите [0-9, A-F].

В RGB-модели любой цвет представлен тремя составляющими или каналами:

* 1. Красным,
  2. Зелёным,
  3. Синим.

Эти цвета называют основными, потому что при их смешивании можно получить любой оттенок.

То же самое можно записать и с использованием альфа-канала:

Вне зависимости от формы записи, альфа-канал является опциональным, и цвет #ff0000 эквивалентен цвету #ff0000ff (красный полностью непрозрачный).

Существует также сокращённая форма шестнадцатеричного представления: на каждый из основных цветов и альфа-канал в таком представлении выделяется по одному символу в шестнадцатеричном алфавите [0-9, A-F]. Такая запись актуальна только для цветов, в которых совпадают первый-второй, третий-четвертый и пятый-шестой символы.

То же самое можно записать и с использованием альфа-канала:

## ****Функциональная запись цветов****

Другая часто встречающаяся форма записи цветов — **функциональная**.В ней каждый цвет разложен по интенсивности тех же основных цветов: красного, зелёного и синего. Параметры записываются через запятую и изменяются от 0 до 255.

То же самое можно записать и с использованием альфа-канала, его значение варьируются между нулём и единицей, где единица означает максимальную степень насыщенности, а ноль — полную прозрачность:

В функциональной форме можно записывать цвета как с использованием специального слова rgb(), так и rgba(). Для создания прозрачности вместо значения единицы в качестве последнего параметра стоит указать степень прозрачности.

Например, параграф с полупрозрачным красным можно задать следующим образом:

color: rgba(255, 0, 0, 0.5);

## ****Запись цветов в системах HSL и HSLA****

Если по работе вы планируете часто общаться с дизайнерами, то имеет смысл получить представление о системе цветового круга, которую они часто используют для подбора контрастов, пар цветов и оттенков.

Выглядит он так:

**Контраст** позволит выбрать два цвета для оформления, **триада** и **тетрада** — три и четыре, соответственно, **аналогия** нужна для поиска комплементарного оттенка, **акцент-аналогия**позволит выбрать и комплементарные оттенки, и контрастный цвет.

Ещё одна система записи цветов — **HSL** — описывает цвета, как раз исходя из положения цвета на цветовом круге.

color: hsl(285, 100%, 60%);

Первый параметр — **оттенок**, который выражен в градусах от 0 до 360, определяющих позицию цвета на цветовом круге.

Второй параметр — **насыщенность**, задаётся в процентах от 0% до 100%.

Третий параметр — **светлота**, который также задаётся в процентах и указывает, насколько светлым или тёмным будет тон цвета.

В современных браузерах rgba() и rgb(), так же как hsl() и hsla(), являются идентичными функциями с различным написанием, то есть значение альфа-каналов можно также писать и в rgb(), и в hsl().

Более подробную спецификацию по типу данных для цвета можно найти [здесь](https://developer.mozilla.org/en-US/docs/Web/CSS/color_value).

## ****Свойства CSS, связанные с цветами****

Свойство [color](https://developer.mozilla.org/en-US/docs/Web/CSS/color" \t "_blank) задаёт цвет элементу:

Свойство [background-color](https://developer.mozilla.org/ru/docs/Web/CSS/background-color" \t "_blank) задаёт фон элемента:

Свойство [border-color](https://developer.mozilla.org/en-US/docs/Web/CSS/border-color" \t "_blank) задаёт цвет границы (но перед тем, как задавать цвет, нужно определиться с её толщиной и дополнительными свойствами):

Свойство [background-image](https://developer.mozilla.org/ru/docs/Web/CSS/background-image" \t "_blank) задаёт фоновую картинку:

### Задание 5.6.1

4 points possible (graded)

* 1. Запишите в шестнадцатеричном представлении цвет с полнонасыщенным зелёным каналом без использования альфа-канала и с нулевой насыщенностью других каналов. В ответе указывайте только значение цвета без селектора и других синтаксических элементов стилей.  нет ответа

#00ff00 **или** #00FF00 **или** #0F0 **или** #0f0

* 1. Запишите в функциональном представлении с использованием цветового пространства RGB полнонасыщенный синий цвет со значением альфа-канала 20%. Формат ответа rgb( ) или rgba( ).  нет ответа

rgb(0, 0, 255, 0.2) **или** rgba(0, 0, 255, 0.2) **или** rgb(0, 0, 255, .2) **или** rgba(0, 0, 255, .2)

* 1. Переведите цвет #37d129 из шестнадцатеричного формата в формат rgb(). Вы можете воспользоваться онлайн-палитрами и другими инструментами. Ответ укажите в формате без точки с запятой. Используйте один пробел после каждой запятой.  нет ответа

rgb(55, 209, 41)

* 1. Какое свойство позволяет задать фоновый цвет элемента? Ответ напишите без тегов и знаков.  нет ответа

## 5.7. Размеры в CSS

## ****Единицы измерения****

**Абсолютные единицы измерения**обозначают физический размер — дюймы, сантиметры, миллиметры, пункты, пики, а также пиксели.

Большинство этих размеров пришли в веб из типографского дела и были очень полезны на начальном этапе развития. Сейчас в основном используются размеры в пикселях (px), но для общего развития можно получить представление о некоторых других.

| **Единица** | **Описание** |
| --- | --- |
| px | Пиксель |
| in | Дюйм (1 дюйм равен 2,54 см) |
| cm | Сантиметр |
| mm | Миллиметр |
| pt | Пункт (1 пункт равен 1/72 дюйма) |
| pc | Пика (1 пика равна 12 пунктам) |

Изначально предполагалось, что пиксель — это самая маленькая часть экрана, один светодиод. Но улучшение мониторов привело к тому, что в экранах некоторых современных компьютеров и смартфонов на один пиксель приходится уже три-четыре светодиода. Это свойство называется **увеличенной плотностью пикселей**и теоретически даёт возможность разработчикам задавать размеры, используя доли пикселей. Например, 12.5px. Но в большинстве случаев браузер округлит это значение. Для мобильных устройств браузер применяет автоматическое масштабирование.

Большинство **относительных единиц измерения** используются для определения размеров шрифта.

|  |  |
| --- | --- |
| **Единица** | **Описание** |
| em | Размер шрифта текущего элемента |
| ex | Высота символа x |
| ch | Ширина символа 0 текущего элемента |
| rem | Размер шрифта корневого элемента |

В каждом браузере заложен размер текста, применяемый в том случае, когда этот размер явно не задан.

Единица **em** — это изменяемое значение, которое зависит от размера шрифта текущего элемента (размер устанавливается через стилевое свойство font-size). Поэтому изначально **1em** равен размеру шрифта, заданному в браузере по умолчанию, или размеру шрифта родительского элемента. **Процентная запись идентична em.**В том смысле, что значения 1em и 100% равны.

Единица **ex** определяется как высота символа «x» в нижнем регистре. На ex распространяются те же правила, что и на em: он привязан к размеру шрифта, заданному в браузере по умолчанию, или к размеру шрифта родительского элемента.

Единица **ch** равна ширине символа «0» для текущего элемента.

Разница между **em** и **rem** следующая: **em** зависит от размера шрифта родителя элемента и меняется вместе с ним, а **rem** привязан к корневому элементу, то есть к размеру шрифта для элемента HTML.

Несмотря на то, что размер шрифта задан одинаково (2em), вторая строчка надписи вдвое больше первой. Всё дело в том, что для параграфа с записью anywhere блок с записью Cats sleep будет родительским, и уже размер этого шрифта будет увеличиваться вдвое.

Одинаковый размер текста в этом случае можно получить при помощи единиц **rem**.

Ещё раз напомним, что запись в процентах будет вести себя аналогично **em**.

Существует также группа относительных единиц измерения, зависящая от **области просмотра браузера**(viewport). Их тоже несколько:

|  |  |
| --- | --- |
| **Единица** | **Описание** |
| vw | 1% от ширины viewport |
| vh | 1% от высоты viewport |
| vmin | 1% от меньшего измерения viewport |
| vmax | 1% от большего измерения viewport |

## ****Базовые свойства, связанные с размерами****

[width](https://developer.mozilla.org/en-US/docs/Web/CSS/width) — ширина. Если раньше задать размер ширины любого блочного элемента в пикселях было нормой, то теперь при вёрстке часто нужно прибегать к свойствам, позволяющим адаптировать размер. Для ширины это:

[min-width](https://developer.mozilla.org/en-US/docs/Web/CSS/min-width) — минимально возможная ширина элемента,

[max-width](https://developer.mozilla.org/en-US/docs/Web/CSS/max-width) — максимально возможная ширина элемента.

[height](https://developer.mozilla.org/en-US/docs/Web/CSS/height) — высота. Свойства, позволяющие адаптировать высоту:

[min-height](https://developer.mozilla.org/en-US/docs/Web/CSS/min-height) — минимально возможная высота элемента,

[max-height](https://developer.mozilla.org/en-US/docs/Web/CSS/max-height) — максимально возможная высота элемента.

[font-size](https://developer.mozilla.org/en-US/docs/Web/CSS/font-size) — размер шрифта.

На свойствах, связанных с размерами отступов и границ, мы остановимся в следующем блоке.

### Задание 5.7.1

1 point possible (graded)

Какие из следующих единиц измерения являются относительными?

px

rem верно

% верно

cm

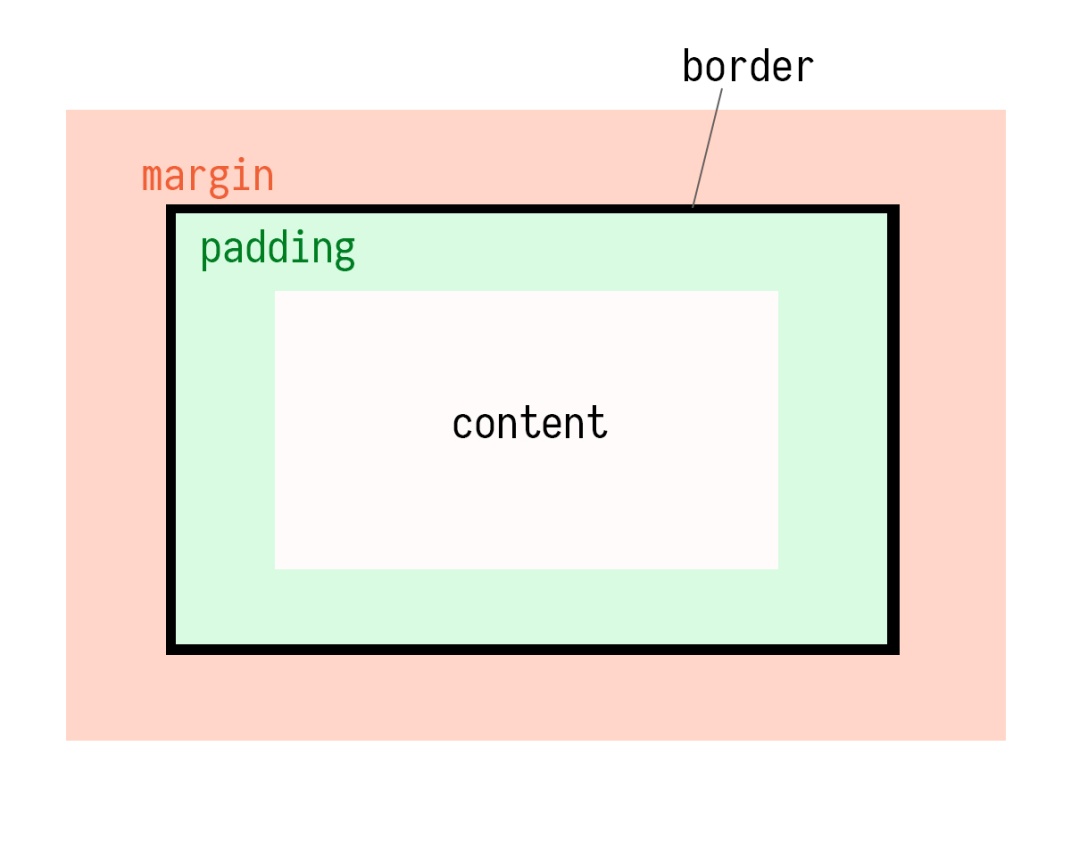
vh верно

**5.8. Блочная модель документа**

 Добавить страницу в мои закладки

Любой блочный элемент можно представить в виде прямоугольного блока, состоящего из набора свойств, которые похожи на слои одежды лютой зимой.

Основой блока выступает его содержимое (текст, картинка, видео), ширина которого задаётся свойством width, а высота — height. Вокруг содержимого идут поля (**padding**), они создают пустое пространство от содержимого до внутреннего края границ; затем идут границы (**border**), а вокруг границ отступы (**margin**). Порядок этих свойств в блоке чётко определён и не может быть нарушен. Но при этом определение ни одного из этих свойств не является обязательным.



## ****Внутренние отступы****

Свойство [padding](https://developer.mozilla.org/ru/docs/Web/CSS/padding" \t "_blank) устанавливает пустое поле между содержимым элемента и его границей. Использование этого свойства повышает читаемость текста, улучшает внешний вид страницы.

У свойства padding может быть до четырёх значений, которые определяют поля с каждой из сторон. Значения перечисляют, начиная с верха, продолжают правым полем, затем — нижним, завершают — левым. То есть **по часовой стрелке**.

padding: верх право низ лево;

padding: 4px 8px 15px 2px;

**Три** значения свойства возможны в том случае, если размеры полей справа и слева совпадают.

padding: верх право-лево низ;

padding: 4px 4px 15px;

**Два** значения задают поля сначала по вертикали, потом по горизонтали.

padding: верх-низ право-лево;

padding: 4px 2px;

Наконец, **одно** значение задаст поля со всех сторон.

Кроме того, для задания полей с одной или двух сторон можно использовать развернутую запись свойства padding:

**padding-top** — изменит поле сверху;

**padding-right** — справа;

**padding-bottom** — снизу;

**padding-left** — слева.

Единицами измерения paddingслужат пиксели, em, rem. Если возникает желание задать внутренние отступы в процентах, нужно учитывать, что и по горизонтали, и по вертикали они будут считаться от **ширины** **всего блока**.

Если задать в качестве значения padding отрицательное число, CSS его проигнорирует.

На практике вам может пригодиться информация о том, что цвет фона, применённый ко всему блоку, останется и после определения внутренних отступов. Значит, если содержимым блока окажется фотография, вы сможете задать ей рамку любого желаемого цвета.

### Задание 5.8.1

1 point possible (graded)

Для следующих значений свойств напишите правило-сокращение с тремя значениями:

padding-right: 10px;

padding-bottom: 16px;

padding-left: 10px;

padding-top: 8px;

Для записи ответа используйте пробел после двоеточия и пробел после каждого значения, то есть можно воспользоваться следующим шаблоном: padding: 0px 0px 0px;  нет ответа

padding: 8px 10px 16px;

Отправить

В некоторых задачах доступны следующие действия: сохранение, сброс, показ подсказки или ответа. Соответствующие кнопки расположены рядом с кнопкой «Отправить».

Показать Ответ

## ****Граница****

Свойство [border](https://developer.mozilla.org/ru/docs/Web/CSS/border" \l "%3Cbr-width%3E" \t "_blank) задаёт вокруг области контента границу определённой толщины, стиля и цвета. Эти значения необходимо последовательно указать через пробел, обязательно нужно задать только стиль.

Можно использовать отдельные свойства [border-width](https://developer.mozilla.org/ru/docs/Web/CSS/border-width" \t "_blank), [border-style](https://developer.mozilla.org/ru/docs/Web/CSS/border-style" \t "_blank), [border-color](https://developer.mozilla.org/ru/docs/Web/CSS/border-color" \t "_blank).

Теперь поговорим о значениях всех трёх свойств.

**Толщину границы**(border-width) можно задать в пикселях или любых других единицах измерения. Но существует также возможность использовать словесное описание. **Thin** — тонкая граница, **medium** — средней толщины, **thick** — толстая. Перевода описания в конкретные единицы измерения не существует. А выглядит каждая из них так:

**Стиль границы**(border-style) может принимать восемь значений. У двух самых востребованных — solid и double — особенностей почти нет. Вид точек и тире в браузерах отличается, поэтому значения **dotted** и **dashed** нужно использовать с осторожностью. Для **groove, ridge, inset** и **outset** лучше задавать два цвета, так как эти значения связаны с переходом цветов.

Обратите внимание:**стиль границы нужно определять обязательно** — либо свойством border-style, либо в общем свойстве border. Иначе она просто не появится.

**Цвет границы** (border-color) можно задавать в любой удобной системе. Обратите внимание, что значение transparent (прозрачный) относится именно к цветам, а не к стилям границы.

Кроме того, как и в случае с padding, есть возможность задать границу только для одной стороны [border-top](https://developer.mozilla.org/en-US/docs/Web/CSS/border-top" \t "_blank), [border-right](https://developer.mozilla.org/en-US/docs/Web/CSS/border-right" \t "_blank), [border-bottom](https://developer.mozilla.org/en-US/docs/Web/CSS/border-bottom" \t "_blank), [border-left](https://developer.mozilla.org/en-US/docs/Web/CSS/border-left" \t "_blank).

К отдельным свойствам границы, которые невозможно задать в сокращённом варианте, относится [border-radius](https://developer.mozilla.org/en-US/docs/Web/CSS/border-radius" \t "_blank). Оно позволяет закруглить края элемента. Это свойство часто используется для рисования в CSS. Вообще, к достоинствам большинства свойств, связанных с border,можно отнести возможность их анимации.

## Круг

## Круг и овал

## ****Свойство margin****

Свойство [margin](https://developer.mozilla.org/en-US/docs/Web/CSS/margin" \t "_blank) устанавливает пустой отступ от внешнего края границы, или внутреннего отступа, или содержимого блока. У margin нет своего фона, поэтому это свойство принимает фон родительского элемента.

Благодаря margin можно задавать отступы от соседних блоков и элементов. По аналогии с padding значения перечисляют, **начиная с верха**, продолжают правым полем, затем — нижним, завершают — левым. То есть **по часовой стрелке**.

margin: верх право низ лево;

margin: 14px 18px 5px 12px;

**Три** значения свойства возможны в том случае, если размеры отступов справа и слева совпадают.

margin: верх право-лево низ;

margin: 4px 4px 15px;

**Два** значения задают отступам сначала по вертикали, потом по горизонтали.

margin: верх-низ право-лево;

margin: 4px 2px;

**Одно** значение margin задаст одинаковые отступы со всех сторон. Точно так же, как и у padding, возможно применение частных свойств: **margin-top**, **margin-right**, **margin-bottom**, **margin-left.**Единицами измерения margin могут быть пиксели, em, rem и так далее. Проценты берутся от ширины всего блока.

Но в отличие от padding:

* + Значение margin может быть отрицательным. То есть, допустим, margin-top: -15px; означает, что блок сдвинется вверх на 15 пикселей, margin-left: 40px; приведёт к перемещению блока влево на 40 пикселей.
  + margin может принимать значение auto. В сочетании с шириной элемента, заданной при помощи свойства width, такое значение позволяет выровнять содержимое элемента по горизонтали.
* **5.9. Слои, позиционирование и z-index**

**Position** — одно из свойств, позволяющих расположить элементы в порядке, отличном от классического потока документа.

Дополнительные свойства, которые определяют расположение элемента:

* + top,
  + right,
  + bottom,
  + left.

Однако дополнительные свойства не будут работать, если не установлено значение position. К значениям относятся:

* + static,
  + relative,
  + fixed,
  + absolute,
  + sticky.

|  |  |
| --- | --- |
| **Position: static;** | Элемент находится в обычном состоянии, расположен на своём месте в документе. Свойства top, right, bottom, left и z-index не применяются к такому элементу. Это значение по умолчанию. |

Элемент считается позиционированным, только если значение position отлично от static.

|  |  |
| --- | --- |
| **Position: relative;** | Элемент с этим значением позиционируется относительно своего нормального положения при помощи свойств top, right, bottom и left. |

|  |  |
| --- | --- |
| **Position: fixed;** | Элемент позиционируется относительно viewport, то есть его положение всегда остаётся неизменным при использовании прокрутки. |

Такое значение часто используется, чтобы зафиксировать верхнюю часть в начале страницы и оставить её доступной постоянно. Также в последнее время часто используют для всплывающего чата в нижней части экрана.

Элемент с position: fixed; не оставляет никакого зарезервированного пространства в том месте, где он определён в документе изначально.

|  |  |
| --- | --- |
| **Position: absolute;** | Элемент позиционируется относительно ближайшего предка (вместо позиционированного относительно окна просмотра, как fixed). Если элемент с абсолютным позиционированием не имеет позиционированных предков, он использует тело документа и перемещается вместе с прокруткой страницы. |

Родителю, относительно которого хочется спозиционировать элемент, часто добавляется свойство position: relative; без указания дополнительных свойств top, right, bottom и left. Таким образом, родительский элемент никак визуально не меняется, а необходимый элемент смещается относительно него.

|  |  |
| --- | --- |
| **Position: sticky;** | Элемент позиционируется на основе позиции прокрутки пользователя. Можно перевести как «липкое» позиционирование. |

Position: sticky; переключает элемент между relative и fixed-позиционированием. Relativeиспользуется, пока положение по умолчанию остаётся во viewport. Когда скролл проходит дальше, включается fixed-позиционирование.

Обратите внимание, что это свойство position **НЕ** поддерживается ни в одной версии браузера Internet Explorer. Подробнее [здесь](https://caniuse.com/#search=sticky).

## ****Слои и z-index****

Когда элементы позиционированы, они могут перекрывать другие элементы, и необходимы какие-то правила, регулирующие это перекрытие.

**Свойство z-index** определяет порядок (какой элемент должен быть расположен сверху или снизу других). Это свойство может иметь как положительные, так и отрицательные значения. Элемент с большим значением перекрывает элемент с меньшим значением.

Если два позиционированных элемента перекрываются без указания z-индекса, элемент, размещённый последним в коде HTML, будет показан сверху.

## Что же регулирует, с точки зрения браузера, z-index?

У каждого элемента в CSS есть определённое место на экране, которое характеризуется координатами X и Y, но если бы существовали только эти две координаты, то мы бы не смогли накладывать один элемент поверх другого. Поэтому существует ещё одна координата — пространственная координата Z. Поэтому z-index так и называется: по сути, это номер слоя, координата по оси Z.

Позиционирование и указание свойства z-index создаёт **контекст наложения** (stacking context).

Задание 5.9.1

1 point possible (graded)

Что называют позиционированным элементом?

Несколько вариантов ответа

Элемент, значение position которого отличается от значения по умолчанию. верно

Элемент, значение position которого отличается от значения static. верно

Элемент, у которого определено свойство position.

Элемент, у которого указан z-index.

нет ответа

Отправить

В некоторых задачах доступны следующие действия: сохранение, сброс, показ подсказки или ответа. Соответствующие кнопки расположены рядом с кнопкой «Отправить».

Показать Ответ

Answers are displayed within the problem

Просмотреть

Задание 5.9.2

1 point possible (graded)

Какое значение position используется элементами по умолчанию?

relative

absolute

fixed

static верно

sticky

none

**5.10 Пример работы со стилями**

До этого момента мы стилизовали текст целыми параграфами. Однако перед разработчиком стоят разные задачи, например, стилизовать в параграфе только одно или несколько слов. Для того чтобы это получилось легко и просто, мы воспользуемся строчным элементом [<span>](https://developer.mozilla.org/en-US/docs/Web/HTML/Element/span). В данном случае этот элемент выступает как обёртка для стилизации части текста, не несущая никакой семантики (смысловой нагрузки). С помощью <span> мы можем оборачивать надписи, ссылки, части текста, чтобы придать им определённый вид.

Возьмём несколько параграфов:

<p>Меня зовут Вася. Я работаю маркетологом. Решил получше узнать смежную профессию, чтобы разобраться, как устроен веб.</p>

<p>Живу в Питере. Люблю кататься на велосипеде.</p>

Допустим, мы хотим стилизовать некоторые слова разным цветом и курсивом. Обернём нужные слова в теги:

<p>Меня зовут <span>Вася</span>. Я работаю <span>маркетологом</span>. Решил получше узнать смежную профессию, чтобы разобраться, как устроен веб.</p>

<p>Живу в <span>Питере</span>. Люблю кататься на велосипеде.</p>

Для того чтобы стилизовать всё по-разному, нам понадобится отличить span друг от друга. Поэтому без **атрибутов** класса элемент span используется редко.

<p>Меня зовут <span class="name-text">Вася</span>. Я работаю <span class="prof-text">маркетологом</span>. Решил получше узнать смежную профессию, чтобы разобраться, как устроен веб.</p>

<p>Живу в <span class="city-text">Питере</span>. Люблю кататься на велосипеде.</p>

Применим разные стили:

**.name-text** {

**font-weight**: **bold**;

}

**.prof-text** {

**color**: **rgb**(**201**, **29**, **29**);

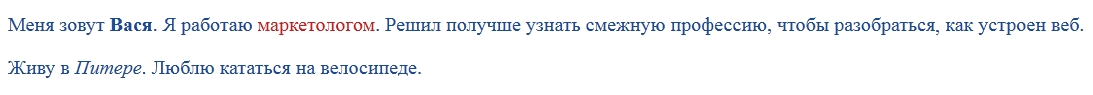
}

**.city-text** {

**font-style**: **italic**;

}

Начертание имени станет жирным, название профессии — красным, а название города — наклонным.



## ****Дополнительные источники информации****

В сети опубликовано огромное количество материалов, посвященных CSS, но официальными ресурсами принято считать только два: [официальную спецификацию CSS](https://www.w3.org/TR/CSS/) и портал [Mozilla Developer Network](https://developer.mozilla.org/ru/" \t "_blank).

В отличие от сухого изложения спецификации, информация на MDN структурирована и изложена в более удобном виде.