**Frontend questions**

Prev

**CSS**

https://www.w3schools.com/css/default.asp

<https://developer.mozilla.org/ru/docs/Learn/CSS/Building_blocks>

CSS (Cascading Style Sheets) — это код, который вы используете для стилизации вашей веб-страницы. Основы CSS помогут вам понять, что вам нужно для начала работы. Мы ответим на такие вопросы как: Как сделать мой текст черным или красным? Как сделать так, чтобы контент появлялся в определённом месте на экране? Как украсить мою веб-страницу с помощью фоновых изображений и цветов?

## [Так что же такое CSS?](https://developer.mozilla.org/ru/docs/Learn/Getting_started_with_the_web/CSS_basics" \l "так_что_же_такое_css)

Как и HTML, CSS на самом деле не является языком программирования. Это не язык разметки - это язык таблицы стилей. Это означает, что он позволяет применять стили выборочно к элементам в документах HTML. Например, чтобы выбрать все элементы абзаца на HTML странице и изменить текст внутри них с чёрного на красный, вы должны написать этот CSS:

p {

color: red;

}

Давайте попробуем: вставьте эти три строки CSS в новый файл в ваш текстовый редактор, а затем сохраните файл как style.css в вашей папке styles.

Но нам всё равно нужно применить CSS к нашему HTML документу. В противном случае, CSS стиль не повлияет на то, как ваш браузер отобразит HTML документ. (Если вы не следили за нашим проектом, то прочитайте раздел [Работа с файлами](https://developer.mozilla.org/ru/docs/Learn/Getting_started_with_the_web/Dealing_with_files) и [Основы HTML](https://developer.mozilla.org/ru/docs/Learn/Getting_started_with_the_web/HTML_basics), чтобы узнать, что вам нужно сделать в первую очередь.)

1. Откройте ваш файл index.html и вставьте следующую строку куда-нибудь в шапку, между <head> и </head> тегами:

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

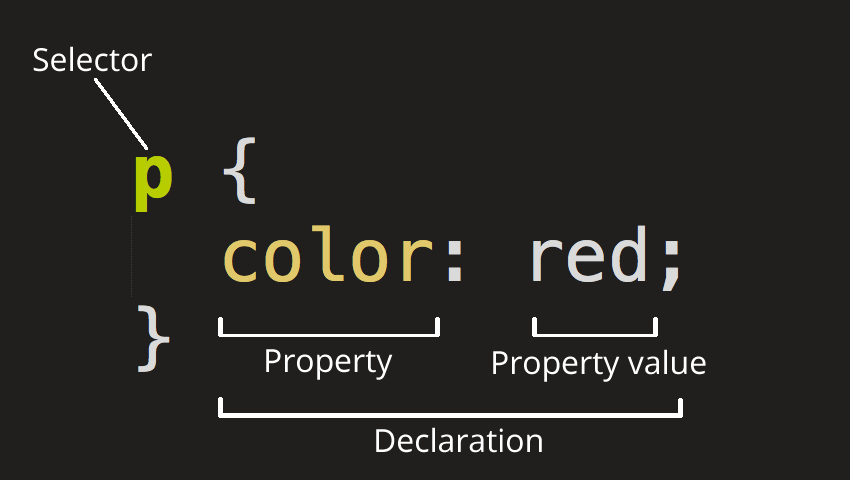
1. Сохраните index.html и загрузите его в вашем браузере. Вы должны увидеть что-то вроде этого:

Copy to Clipboard

Если текст вашего абзаца теперь красный, примите наши поздравления! Вы написали свой первый успешный CSS!

### [Анатомия набора правил CSS](https://developer.mozilla.org/ru/docs/Learn/Getting_started_with_the_web/CSS_basics" \l "анатомия_набора_правил_css)

Давайте взглянем на вышеупомянутый CSS немного более подробно:



Вся структура называется набором правил (но зачастую для краткости "правило"). Отметим также имена отдельных частей:

Селектор (Selector)

Имя HTML-элемента в начале набора правил. Он выбирает элемент(ы) для применения стиля (в данном случае, элементы p ). Для стилизации другого элемента, просто измените селектор.

Объявление (Declaration)

Единственное правило, например color: red; указывает, какие из свойств элемента вы хотите стилизовать.

Свойства (Properties)

Способы, которыми вы можете стилизовать определённый HTML-элемент (в данном случае, color является свойством для элементов [<p>](https://developer.mozilla.org/ru/docs/Web/HTML/Element/p)). В CSS вы выбираете, какие свойства вы хотите затронуть в вашем правиле.

Значение свойства (Property value)

Справа от свойства, после двоеточия, у нас есть значение свойства, которое выбирает одно из множества возможных признаков для данного свойства (существует множество значений color, помимо red).

Обратите внимание на важные части синтаксиса:

* Каждый набор правил (кроме селектора) должен быть обёрнут в фигурные скобки ({}).
* В каждом объявлении необходимо использовать двоеточие (:), чтобы отделить свойство от его значений.
* В каждом наборе правил вы должны использовать точку с запятой (;), чтобы отделить каждое объявление от следующего.

Таким образом, чтобы изменить несколько значений свойств сразу, вам просто нужно написать их, разделяя точкой с запятой, например так:

p {

color: red;

width: 500px;

border: 1px solid black;

}

### Выбор нескольких элементов

[Copy to Clipboard](https://developer.mozilla.org/ru/docs/Learn/Getting_started_with_the_web/CSS_basics" \l "выбор_нескольких_элементов)

Вы также можете выбрать несколько элементов разного типа и применить единый набор правил для всех из них. Добавьте несколько селекторов, разделённых запятыми. Например:

p,li,h1 {

color: red;

}

### Разные типы селекторов

[Copy to Clipboard](https://developer.mozilla.org/ru/docs/Learn/Getting_started_with_the_web/CSS_basics" \l "разные_типы_селекторов)

Существует множество различных типов селектора. Выше мы рассматривали только селектор элементов, который выбирает все элементы определённого типа в HTML документе. Но мы можем сделать выбор более конкретным. Вот некоторые из наиболее распространённых типов селекторов:

| Имя селектора | Что выбирает | Пример |
| --- | --- | --- |
| Селектор элемента (иногда называемый селектором тега или типа) | Все HTML-элемент(ы) указанного типа. | p Выбирает <p> |
| ID селектор | Элемент на странице с указанным ID на данной HTML. Лучше всего использовать один элемент для каждого ID (и конечно один ID для каждого элемента), даже если вам разрешено использовать один и тот же ID для нескольких элементов. | #my-id Выбирает <p id="my-id"> или <a id="my-id"> |
| Селектор класса | Элемент(ы) на странице с указанным классом (множество экземпляров класса может объявляться на странице). | .my-class Выбирает <p class="my-class"> и <a class="my-class"> |
| Селектор атрибута | Элемент(ы) на странице с указанным атрибутом. | img[src] Выбирает <img src="myimage.png"> но не <img> |
| Селектор псевдокласса | Указанные элемент(ы), но только в случае определённого состояния, например, при наведении курсора. | a:hover Выбирает <a>, но только тогда, когда указатель мыши наведён на ссылку. |

Существует ещё много селекторов для изучения, и вы можете найти более подробный список в нашем [Руководстве селекторов (en-US)](https://developer.mozilla.org/en-US/docs/Learn/CSS/Building_blocks/Selectors).

## [Шрифты и текст](https://developer.mozilla.org/ru/docs/Learn/Getting_started_with_the_web/CSS_basics" \l "шрифты_и_текст)

Теперь, когда мы изучили некоторые основы CSS, давайте добавим ещё несколько правил и информацию в наш файл style.css, чтобы наш пример хорошо выглядел. Прежде всего, давайте сделаем, чтобы наши шрифты и текст выглядели немного лучше.

1. Прежде всего, вернитесь и найдите [вывод из Google Fonts](https://developer.mozilla.org/ru/docs/Learn/Getting_started_with_the_web/What_will_your_website_look_like), который вы уже где-то сохранили. Добавьте элемент [<link>](https://developer.mozilla.org/ru/docs/Web/HTML/Element/link) где-нибудь внутри шапки вашего index.html (снова, в любом месте между тегами <head> и </head>). Это будет выглядеть примерно так:

<link href='http://fonts.googleapis.com/css?family=Open+Sans' rel='stylesheet' type='text/css'>

Этот код связывает вашу страницу с таблицой стилями, которая загружает семейство шрифтов Open Sans вместе с вашей страницей и позволяет вам применять их к вашим HTML-элементам используя свою собственную таблицу стилей.

Copy to Clipboard

1. Затем, удалите существующее правило в вашем style.css файле. Это был хороший тест, но красный текст, на самом деле, не очень хорошо выглядит.
2. Добавьте следующие строки в нужное место, заменив строку placeholder актуальной font-family строкой, которую вы получили из Google Fonts. (font-family просто означает, какой шрифт(ы) вы хотите использовать для вашего текста). Это правило устанавливает глобальный базовый шрифт и размер шрифта для всей страницы (поскольку <html> является родительским элементом для всей страницы, и все элементы внутри него наследуют такой же font-size и font-family):

html {

font-size: 10px; /\* px значит 'пиксели': базовый шрифт будет 10 пикселей в высоту \*/

font-family: placeholder: здесь должно быть имя шрифта из Google fonts

}

Примечание: Все в CSS документе между /\* и \*/ является CSS комментарием, который браузер игнорирует при исполнении кода. Это место, где вы можете написать полезные заметки о том, что вы делаете.

1. Теперь мы установим размер шрифта для элементов, содержащих текст внутри HTML тела ([<h1> (en-US)](https://developer.mozilla.org/en-US/docs/Web/HTML/Element/Heading_Elements), [<li>](https://developer.mozilla.org/ru/docs/Web/HTML/Element/li), и [<p>](https://developer.mozilla.org/ru/docs/Web/HTML/Element/p)). Мы также отцентрируем текст нашего заголовка и установим некоторую высоту строки и расстояние между буквами в теле документа, чтобы сделать его немного более удобным для чтения:

Copy to Clipboard

h1 {

font-size: 60px;

text-align: center;

}

p, li {

font-size: 16px;

line-height: 2;

letter-spacing: 1px;

}

Copy to Clipboard

Вы можете настроить значения px так, как вам нравится, чтобы ваш дизайн выглядел так, как вы хотите, но, в общем, ваш дизайн должен выглядеть вот так:

## [Блоки, блоки и ещё раз блоки](https://developer.mozilla.org/ru/docs/Learn/Getting_started_with_the_web/CSS_basics" \l "блоки_блоки_и_ещё_раз_блоки)

Одна вещь, которую вы заметите в написании CSS, заключается в том, что многое из этого касается блоков - настройка их размера, цвета, положения и т.д. Большинство HTML-элементов на странице можно рассматривать как блоки, расположенные друг над другом.

Не удивительно, макет CSS основан, главным образом, на блочной модели (box model). Каждый из блоков, занимающий пространство на вашей странице имеет такие свойства, как:

* padding, пространство только вокруг контента (например, вокруг абзаца текста)
* border, сплошная линия, которая расположена рядом с padding
* margin, пространство вокруг внешней стороны элемента

В этом разделе мы также используем:

* width (ширину элемента)
* background-color, цвет позади контента и padding элементов
* color, цвет контента элемента (обычно текста)
* text-shadow: устанавливает тень на тексте внутри элемента
* display: устанавливает режим отображения элемента (пока что не волнуйтесь об этом)

Итак, давайте начнём и добавим больше CSS на нашей странице! Продолжайте добавлять эти новые правила, расположенные в нижней части страницы, и не бойтесь экспериментировать с изменением значений, чтобы увидеть, как это работает.

### [Изменение цвета страницы](https://developer.mozilla.org/ru/docs/Learn/Getting_started_with_the_web/CSS_basics" \l "изменение_цвета_страницы)

html {

background-color: #00539F;

}

Это правило устанавливает цвет фона для всей страницы. Измените код цвета сверху, на цвет который [вы выбрали при планировании вашего сайта](https://developer.mozilla.org/ru/docs/Learn/Getting_started_with_the_web/What_will_your_website_look_like).

Copy to Clipboard

### [Разбираемся с телом](https://developer.mozilla.org/ru/docs/Learn/Getting_started_with_the_web/CSS_basics" \l "разбираемся_с_телом)

body {

width: 600px;

margin: 0 auto;

background-color: #FF9500;

padding: 0 20px 20px 20px;

border: 5px solid black;

}

Теперь для [<body>](https://developer.mozilla.org/ru/docs/Web/HTML/Element/body) элемента. Здесь есть немало деклараций, так что давайте пройдём через них всех по одному:

Copy to Clipboard

* width: 600px; — заставляет тело быть всегда 600 пикселей в ширину.
* margin: 0 auto; — когда вы устанавливаете два значения для таких свойств как margin или padding, первое значение элемента влияет на верхнюю и нижнюю сторону (делает их 0 в данном случае), и второе значение на левую и правую сторону (здесь, auto является особым значением, которое делит доступное пространство по горизонтали поровну слева и справа). Вы также можете использовать один, три или четыре значения, как описано [здесь](https://developer.mozilla.org/ru/docs/Web/CSS/margin" \l "values).
* background-color: #FF9500; — как и прежде, устанавливает цвет фона элемента. Я использовал красновато-оранжевый для тела, в отличие от темно-синего цвета для [<html>](https://developer.mozilla.org/ru/docs/Web/HTML/Element/html) элемента, но не стесняйтесь и экспериментируйте.
* padding: 0 20px 20px 20px; — у нас есть четыре значения, установленные для padding, чтобы сделать немного пространства вокруг нашего контента. В этот раз мы не устанавливаем padding на верхней части тела, но делаем 20 пикселей слева, снизу и справа. Значения устанавливаются сверху, справа, снизу, слева, в таком порядке.
* border: 5px solid black; — просто устанавливает сплошную чёрную рамку шириной 5 пикселей со всех сторон тела.

### [Позиционирование и стилизация нашего заголовка главной страницы](https://developer.mozilla.org/ru/docs/Learn/Getting_started_with_the_web/CSS_basics" \l "позиционирование_и_стилизация_нашего_заголовка_главной_страницы)

h1 {

margin: 0;

padding: 20px 0;

color: #00539F;

text-shadow: 3px 3px 1px black;

}

Вы, возможно, заметили, что есть ужасный разрыв в верхней части тела. Это происходит, потому что браузеры применяют некоторый стиль по умолчанию для элемента [<h1> (en-US)](https://developer.mozilla.org/en-US/docs/Web/HTML/Element/Heading_Elements) (по сравнению с другими), даже если вы не применяли какой-либо CSS вообще! Это может звучать как плохая идея, но мы хотим, чтобы веб-страница без стилей имела базовую читаемость. Чтобы избавиться от разрыва, мы переопределили стиль по умолчанию, установив margin: 0;.

Copy to Clipboard

Затем мы установили заголовку верхний и нижний padding на 20 пикселей, и сделали текст заголовка того же цвета, как и цвет фона html.

Здесь, мы использовали одно довольно интересное свойство - это text-shadow, которое применяет тень к текстовому контенту элемента. Оно имеет следующие четыре значения:

* Первое значение пикселей задаёт горизонтальное смещение тени от текста — как далеко она движется поперёк: отрицательное значение должно двигать её влево.
* Второе значение пикселей задаёт вертикальное смещение тени от текста — как далеко она движется вниз, в этом примере: отрицательное значение должно переместить её вверх.
* Третье значение пикселей задаёт радиус размытия тени — большее значение будет означать более размытую тень.
* Четвёртое значение задаёт основной цвет тени.

И вновь попробуйте поэкспериментировать с различными значениями, чтобы посмотреть, что вы можете придумать.

### [Центрирование изображения](https://developer.mozilla.org/ru/docs/Learn/Getting_started_with_the_web/CSS_basics" \l "центрирование_изображения)

img {

display: block;

margin: 0 auto;

}

В заключение, мы отцентрируем изображение, чтобы оно лучше выглядело. Мы можем использовать margin: 0 auto уловку снова, как мы это делали раньше для body, но мы также должны сделать кое-что ещё. Элемент [<body>](https://developer.mozilla.org/ru/docs/Web/HTML/Element/body) является блочным, это значит, что он занимает всю ширину страницы и может иметь margin и другие значения отступов, применяемых к нему. Изображения, наоборот, являются строчными элементами, то есть они этого не могут. Таким образом, чтобы применить margin к изображению, мы должны дать изображению блочное поведение с помощью display: block;.

Copy to Clipboard

Примечание: Приведённые выше инструкции предполагают, что вы используете изображение меньшей ширины, чем заданная для [<body>](https://developer.mozilla.org/ru/docs/Web/HTML/Element/body) (600px). Если ваше изображение больше, тогда оно выйдет за границы [<body>](https://developer.mozilla.org/ru/docs/Web/HTML/Element/body) и займёт пространство страницы. Чтобы исправить это, вы можете: 1) уменьши ширину изображения используя [графический редактор](https://ru.wikipedia.org/wiki/Растровый_графический_редактор), или 2) изменить размер изображения используя CSS путём установки свойства [width](https://developer.mozilla.org/ru/docs/Web/CSS/width) для <img> меньше, чем у [<body>](https://developer.mozilla.org/ru/docs/Web/HTML/Element/body).

Примечание: Не стоит беспокоиться, если вы ещё не понимаете display: block; и различия между блочным/строчным. Вы поймёте, когда будете изучать CSS более подробно. Вы можете узнать больше о различных доступных значениях display на нашей странице о свойстве [display](https://developer.mozilla.org/ru/docs/Web/CSS/display).

[**Варианты добавление CSS стилей на страницу?**](https://youtu.be/ycYp7CYOnO0?t=277)

## Three Ways to Insert CSS

There are three ways of inserting a style sheet:

* External CSS
* Internal CSS
* Inline CSS

## External CSS

With an external style sheet, you can change the look of an entire website by changing just one file!

Each HTML page must include a reference to the external style sheet file inside the <link> element, inside the head section.

### Example

External styles are defined within the <link> element, inside the <head> section of an HTML page:

<!DOCTYPE html>  
<html>  
<head>  
<link rel="stylesheet" href="mystyle.css">  
</head>  
<body>  
  
<h1>This is a heading</h1>  
<p>This is a paragraph.</p>  
  
</body>  
</html>

An external style sheet can be written in any text editor, and must be saved with a .css extension.

The external .css file should not contain any HTML tags.

## Internal CSS

An internal style sheet may be used if one single HTML page has a unique style.

The internal style is defined inside the <style> element, inside the head section.

### Example

Internal styles are defined within the <style> element, inside the <head> section of an HTML page:

<!DOCTYPE html>  
<html>  
<head>  
<style>  
body {  
  background-color: linen;  
}  
  
h1 {  
  color: maroon;  
  margin-left: 40px;  
}  
</style>  
</head>  
<body>  
  
<h1>This is a heading</h1>  
<p>This is a paragraph.</p>  
  
</body>  
</html

## Inline CSS

An inline style may be used to apply a unique style for a single element.

To use inline styles, add the style attribute to the relevant element. The style attribute can contain any CSS property.

### Example

Inline styles are defined within the "style" attribute of the relevant element:

<!DOCTYPE html>  
<html>  
<body>  
  
<h1 style="color:blue;text-align:center;">This is a heading</h1>  
<p style="color:red;">This is a paragraph.</p>  
  
</body>  
</html>

**Tip:** An inline style loses many of the advantages of a style sheet (by mixing content with presentation). Use this method sparingly.

## Multiple Style Sheets

If some properties have been defined for the same selector (element) in different style sheets, the value from the last read style sheet will be used.

Assume that an **external style sheet** has the following style for the <h1> element:

h1 {  
  color: navy;  
}

Then, assume that an **internal style sheet** also has the following style for the <h1> element:

h1 {  
  color: orange;     
}

### Example

If the internal style is defined **after** the link to the external style sheet, the <h1> elements will be "orange":

<head>  
<link rel="stylesheet" type="text/css" href="mystyle.css">  
<style>  
h1 {  
  color: orange;  
}  
</style>  
</head>

### Example

However, if the internal style is defined **before** the link to the external style sheet, the <h1> elements will be "navy":

<head>  
<style>  
h1 {  
  color: orange;  
}  
</style>  
<link rel="stylesheet" type="text/css" href="mystyle.css">  
</head>

## Cascading Order

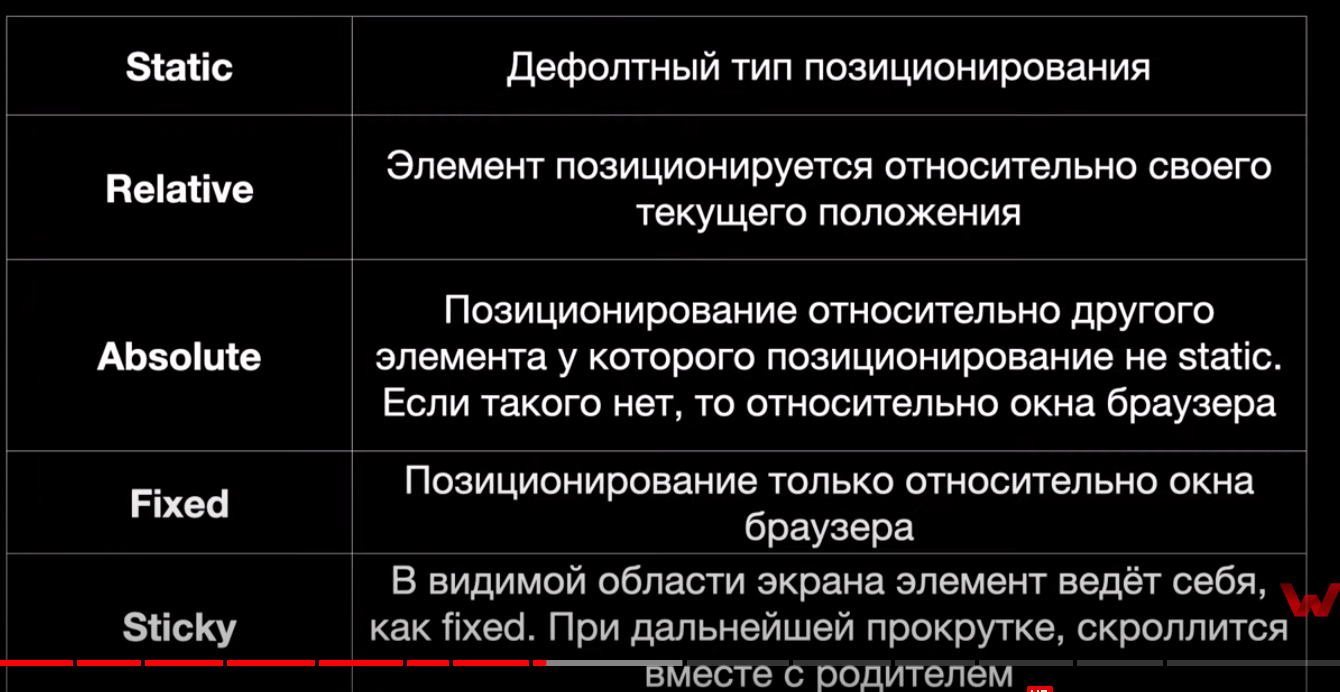
What style will be used when there is more than one style specified for an HTML element?

All the styles in a page will "cascade" into a new "virtual" style sheet by the following rules, where number one has the highest priority:

1. Inline style (inside an HTML element)
2. External and internal style sheets (in the head section)
3. Browser default

So, an inline style has the highest priority, and will override external and internal styles and browser defaults.

[Типы позиционирования в CSS?](https://youtu.be/ycYp7CYOnO0?t=321)

****

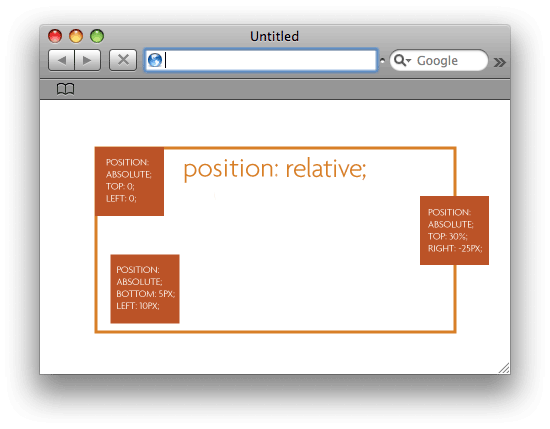
При работе с позиционированием важно понять одну концепцию: все элементы на странице - это блоки. Грубо говоря, прямоугольники из пикселей. Это означает, что каждому блоку можно присвоить и высоту, и ширину в пикселях. Блоки (элементы) бывают также инлайновыми. Они тоже в виде прямоугольников, но лежат на странице немного иначе. Они выстраиваются друг за другом по-горизонтали.

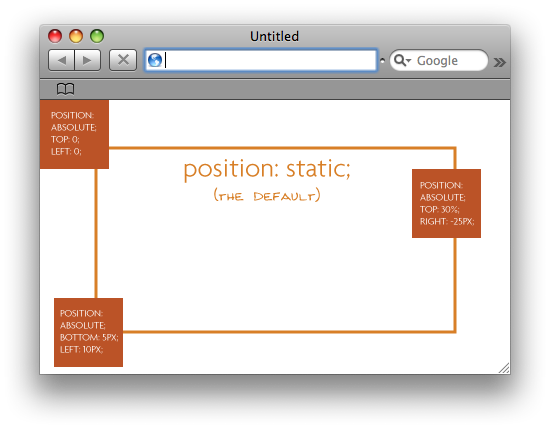
Теперь, когда мы для себя решили, что каждый элемент на странице это блок, мы можем поговорить о типах позиционирования и как они помогут нам добиться нужного результата (т.е. разместить любой блок там, где нам необходимо).

- **STATIC.**Это значение по умолчанию для любого элемента на странице. Это самый простой тип и практически ничего не означает. Все выглядит как должно выглядеть.

- **RELATIVE.** Этот тип позиционирования наиболее трудный и неправильно используемый. Если ВЫ присвоете элементу position: relative;, но не придадите каких-либо других атрибутов (top, left, bottom или right), тогда ничего не изменится. Все останется на своих местах (также как и первым типом позиционирования). Но если Вы добавите атрибуты (например, top: 10px), тогда элемент сместится на 10 пикселей вниз от того места, где он обычно располагался. Это очень полезно при выравнивании элементов формы в один ряд.

Кроме этого, есть еще две вещи, которые происходят при употреблении position: relative. Во-первых, это возможность использовать на этом элементе z-index. С помощью этого свойства можно добиться интересных решений для любого сайта. Во-вторых, Все дочерние элементы относительно позиционированого элемента могут быть позиционированы абсолютно. Это позволяет решать практически любые задачи по оформлению страничек. Если не совсем понятна суть, тогда достаточно взглянуть на изображения ниже:





- **ABSOLUTE.** Очень мощный тип позиционирования, который позволяет поместить любой элемент в любом месте страницы. Для этого используются аргументы top, left, bottom и right. Самое главное запомнить, что элементы с абсолютным позиционированием как бы вырваны из структуры страницы. На элемент с таким типом позиционирования не влияют другие элементы, и он также не влияет на них. Это необходимо всегда держать в уме при создании сайтов. Неправильное или частое использование такого типа может ограничить гибкость Вашего сайта.

- **FIXED.** Этот тип позиционирования очень редкий. Элемент с таким типом позиционируется относительно к окну просмотра, то есть браузеру. При скролле, такой элемент будет оставаться всегда на том же месте. Взгляните на [этот сайт](http://www.bluelounge.com/index.php). При прокрутке вниз с левой стороны меню остается на одном и том же месте. Этот пример одновременно показывает и плюсы, и минусы такого типа позиционирования. Преимуществом является то, что навигация всегда на виду, и вообще это интересный эффект для посетителей. Минусы возможно не видны с первого взгляда. Если зайти на эту страницу с ноутбука с маленьким экраном, например, тогда панель с навигацией будет видна не полностью (и никак не удасться ее просмотреть, так как не будет скролла). В общем, это классный эффект, но требует тщательного тестирования.

## Базовый поток документа

HTML-документ состоит из большого количества элементов, вложенных друг в друга. Чтобы из этих элементов и CSS **построить изображение страницы**, их необходимо как-то в ней расположить. По умолчанию размещение всех элементов на странице осуществляется в **нормальном или базовом потоке**.

**Что это значит?** Во-первых, вывод элементов на страницу браузер осуществляет **в том порядке, в котором они следуют в HTML коде**.

Во-вторых, в коде элементы вложены друг в друга, и чтобы это учитывать при выводе используют так называемые **воображаемые слои для отображения элементов**. При этом слой элемента тем выше (ближе к нам), чем данный элемент является более вложенным в коде, т.е. глубже расположен в нём.

HTMLКопировать

<body>

Этот элемент находится позади других элементов.

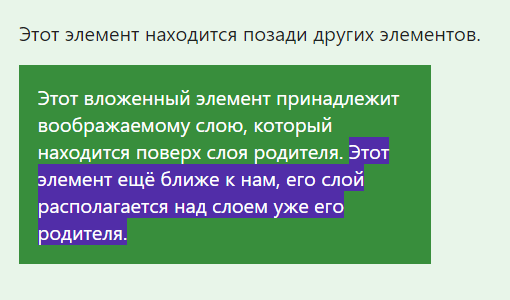
<div>

Этот вложенный элемент принадлежит воображаемому слою, который находится поверх слоя родителя.

<span>Этот элемент ещё ближе к нам, его слой располагается над слоем уже его родителя.</span>

</div>

</body>



В-третьих, положение элемента в потоке зависит от значения свойства display.

HTMLКопировать

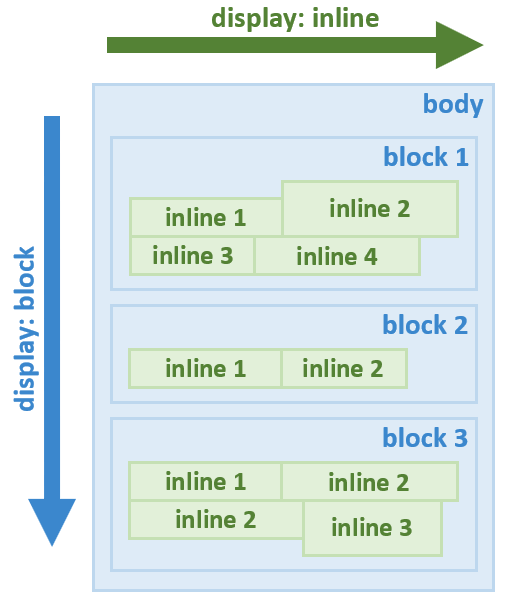
<body>

<div class="block-1"><span class="inline-1">inline 1</span><span class="inline-2">inline 2</span><span class="inline-3">inline 3</span><span class="inline-4">inline 4</span></div>

<div class="block-2"><span class="inline-1">inline 1</span><span class="inline-2">inline 2</span></div>

<div class="block-3"><span class="inline-1">inline 1</span><span class="inline-2">inline 2</span><span class="inline-3">inline 3</span></div>

</body>



Например, элементы, имеющее **блочное отображение** (display: block) отображаются в потоке как прямоугольные области, каждый из них на новой линии друг под другом сверху вниз.

Ширина элементов с блочным отображением по умолчанию равна доступной ширине родительского элемента, т.е. элемента, в который каждый из них непосредственно вложен. Высота их по умолчанию равна такой величине, которой будет достаточно, чтобы отобразить весь контент, который находится в каждом из них.

Элементы со **строчным отображением** (display: inline) выводятся иначе. Они в отличии от блочных элементов не размещаются каждый на новой строке, а следуют друг за другом слево направо. Если пространство справа закончилось, то они переносятся на следующую строку, а не на новую линию как элементы с блочным отображением.

Кроме block, inline есть и другие варианты отображения элементов, но все они располагаются в базовом потоке документа.

**В CSS есть свойства, с помощью которых элементы можно «вырвать» из основного потока документа и задать им другое положение вне базового потока элементов.**

К этим свойствам относятся position и float.

## CSS-свойство position

CSS свойство position — это одно из свойств с помощью которого можно изменить базовое поведение элементов в потоке. Другими словами, данное свойство позволяет «выдернуть» любой элемент из потока документа и разместить его в другом месте относительно окна браузера или других элементов на веб-странице.

Свойство position имеет 5 значений:

* static (статичное позиционирование);
* relative (относительное);
* absolute (абсолютное);
* fixed (фиксированное);
* sticky (липкое).

static — это значение по умолчанию. Оно означает что элемент находится в базовом потоке.

Каждый элемент в потоке занимает определённую область. Но область элемента не всегда сохраняется за ним при его позиционировании.

Это, например, происходит при задании элементу position: absolute или position: fixed. В этом случае место не сохраняется за элементом. Другие элементы его «не видят» и располагаются, игнорируя его присутствие в коде.

## Статичное позиционирование (static)

Свойство position со значением static элементам назначается **по умолчанию**. Это значение означает что элемент является **не позиционированным**, т.е. отображается как обычно (в потоке).

Явная установка элементу CSS-свойства position: static может понадобиться только в том случае, когда нужно переопределить другое значение position установленное элементу.

Установка CSS свойств для задания положения элемента left, top, right и bottom никакого влияния на него не оказывают, т.к. его **местонахождение определяется потоком документа**.

Пример выстраивания статично позиционированных элементов:

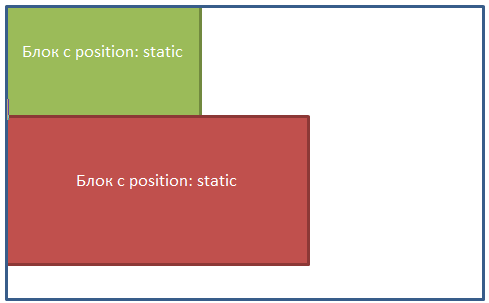
HTMLКопировать

<body>

<div style="width: 200px; height: 100px; border: 1px solid black; background: green;"></div>

<div style="width: 300px; height: 150px; border: 1px solid black; background: red;"></div>

</body>



## Относительное позиционирование (relative)

Установка относительного позиционирования элементу осуществляется посредством задания ему CSS свойства position: relative.

Относительно позиционированный элемент ведёт себя как элемент в потоке за исключением того, что его текущее положение можно при помощи определённых CSS свойств сместить. К этим CSS свойствам относятся left, top, right и bottom.

Например, для того чтобы элемент сдвинуть вверх или вниз относительного его исходного положения к нему нужно применить CSS свойство top или bottom:

CSSКопировать

position: relative;

/\* для сдвига элемента вверх на 10px \*/

top: -10px; /\* или bottom: 10px; \*/

/\* для сдвига элемента вниз на 10px \*/

top: 10px; /\* или bottom: -10px; \*/

Если одновременно установить top и bottom, то будет применено значение top, т.к. оно является более приоритетным, чем bottom:

CSSКопировать

position: relative;

/\* элемент или элементы, к которым применяется эти стили будут сдвинуты на 15px вверх, а не на 10px как указано в bottom \*/

top: -15px;

bottom: 10px;

Для сдвига элемента вправо или влево используется CSS свойство left или right:

CSSКопировать

position: relative;

/\* для сдвига элемента влево на 20px \*/

left: -20px; /\* или right: 20px; \*/

/\* для сдвига элемента вправо на 20px \*/

left: 20px; /\* или right: -20px; \*/

Если одновременно установить left и right, то приоритетным будет значение, находящееся в left:

CSSКопировать

position: relative;

/\* элемент или элементы, к которым применяется эти стили будут сдвинуты на 25px вправо, т.к. значение left более приоритетно чем right \*/

left: 25px;

right: -20px;

Для сдвига по двум осям нужно использовать top или bottom, и left или right:

CSSКопировать

position: relative;

/\* стили для сдвига элементов вверх и влево на 5px \*/

top: -5px;

left: -5px;

Пример, в котором 2 элементу установим относительное позиционирование и сместим его на 20px вверх и влево относительно его исходного положения:

HTMLКопировать[Запустить](https://itchief.ru/examples/lab.php?topic=html-and-css&file=css-positioning)

<div class="container">

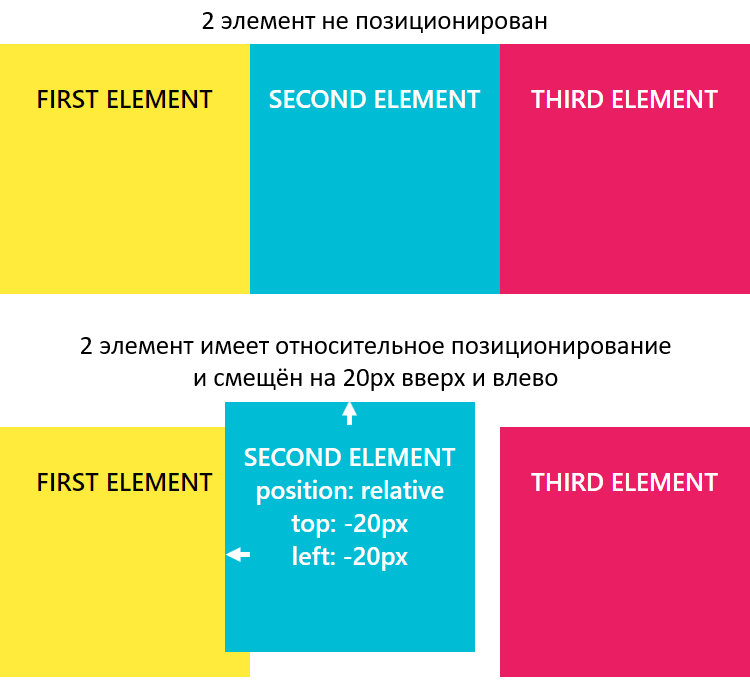
<div class="element-1">FIRST ELEMENT</div>

<!-- элемент имеет относительное позиционирование и смещён на 20px вверх и влево -->

<div class="element-2" style="position: relative; top: -20px; left: -20px;">SECOND ELEMENT</div>

<div class="element-3">THIRD ELEMENT</div>

</div>



Если в некоторой области страницы оказываются несколько позиционированных элементов, то они перекрывают друг на друга в определённом порядке. При этом по умолчанию выше оказывается тот элемент, который ниже описан в коде. Но порядок перекрытия элементов (их положение перпендикулярное экрану, т.е. вдоль оси Z) можно изменить. Осуществляется в CSS это с помощью свойства z-index. z-index может принимать отрицательные и положительные целые число, auto и 0. Но, хорошей практикой является использование в качестве z-index чисел из диапазона 0-9999.

При этом чем больше у элемента значение z-index, тем ближе он располагается к нам, и, следовательно, перекрывает все элементы в данной области, у которых значение z-index меньше.

## Абсолютное позиционирование (absolute)

Установка абсолютного позиционирования элементу осуществляется посредством задания ему position: absolute.

Этот тип позиционирования позволяет разместить элемент именно там, где вы хотите.

Позиционирование выполняется относительно ближайшего позиционированного предка.

HTMLКопировать

<div id="id-1" style="position: absolute">

<div id="id-2" style="position: relative">

<div id="id-3" style="position: absolute">

...

</div>

</div>

</div>

Под позиционированным элементом понимается элемент с position, равным relative, absolute, fixed или sticky.

В этом примере позиционирование элемента #id-3 будет выполнять относительно #id-2, т.к. он является позиционированным и является по отношению к нему более близким предком.

Если данный элемент не был бы позиционированным, то позиционирование #id-3 выполнялось бы относительно #id-1:

HTMLКопировать

<div id="id-1" style="position: absolute">

<div id="id-2">

<div id="id-3" style="position: absolute">

...

</div>

</div>

</div>

Если среди предков у элемента с position: absolute нет позиционированного элемента, то в этом случае он будет позиционироваться относительно HTML страницы, т.е. элемента body.

Когда элементу устанавливаем position: absolute без указания CSS-свойств (top, left, right и bottom), определяющих его положение, он будет находиться в том месте, в котором он был бы расположен, если бы находился в потоке (при этом при вычислении его положения учитываются только элементы, расположенные до него в коде и находящиеся в потоке).

При этом другие элементы его видеть не будут, и, следовательно, они будут расположены на странице, не обращая никакого внимание на него.

HTMLКопировать[Запустить](https://itchief.ru/examples/lab.php?topic=html-and-css&file=css-positioning-2)

<div class="container">

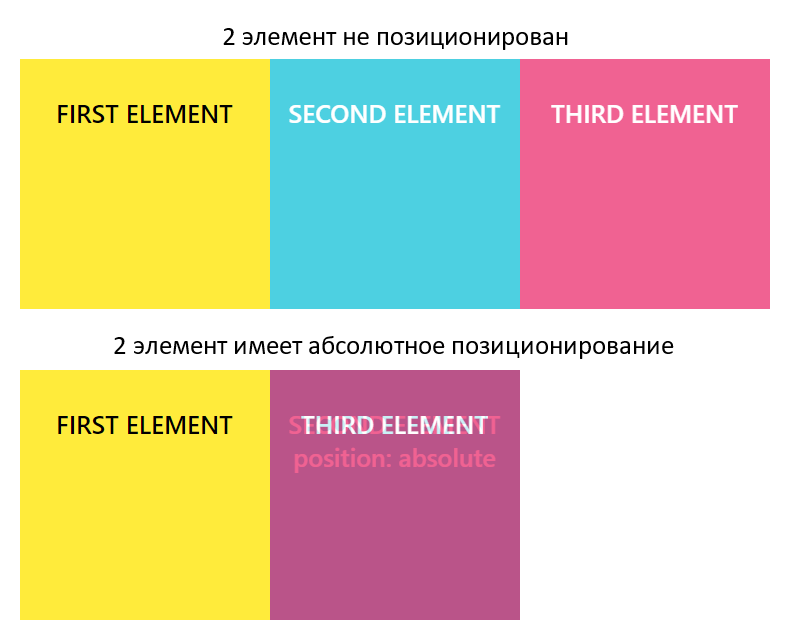
<div class="element-1">FIRST ELEMENT</div>

<!-- элемент имеет абсолютное позиционирование и ему не установлены CSS-свойства top, bottom, left и right -->

<div class="element-2" style="position: absolute;">SECOND ELEMENT</div>

<div class="element-3">THIRD ELEMENT</div>

</div>



CSS-свойства для управления положением абсолютно позиционированного элемента работают по-другому чем с position: relative.

CSS-свойства top, bottom, left и right задают положение элемента относительно ближайшего позиционированного предка или body, если такого предка нет.

Установить ширину (высоту) абсолютно позиционированному можно с помощью установки ему двух координат top и bottom (left и right).

Если элементу одновременно установить top, bottom и height, то предпочтение будет отдано top и height.

Абсолютное позиционирование применяется очень часто совместно с относительным позиционированием в дизайнерских целях, когда необходимо разместить различные элементы относительного друг друга, так же может применяться для создания выпадающих меню, разметки сайта и т.д.

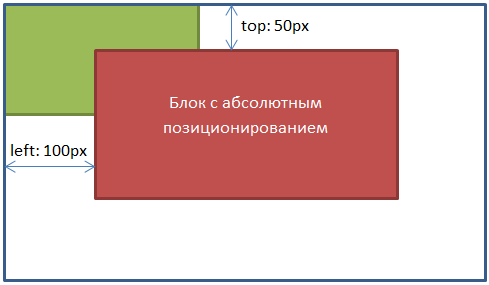
HTMLКопировать

<body>

<div style="width: 200px; height: 100px; border: 1px solid black; background: green;"></div>

<div style="width: 300px; height: 200px; position: absolute; top: 50px; left: 100px; border: 1px solid black; background:red;"></div>

</body>



## Фиксированное позиционирование (fixed)

Задание элементу фиксированного позиционирования осуществляется посредством установки ему position: fixed.

Фиксированное позиционирование похоже на абсолютное, но в отличии от него оно **всегда привязывается к краям окна браузера** (viewport), и **остаётся в таком положении даже при скроллинге страницы**.

Фиксированное позиционирование применяется для закрепления на странице навигационных меню, кнопки «вверх», панелей с социальными кнопками и многого другого.

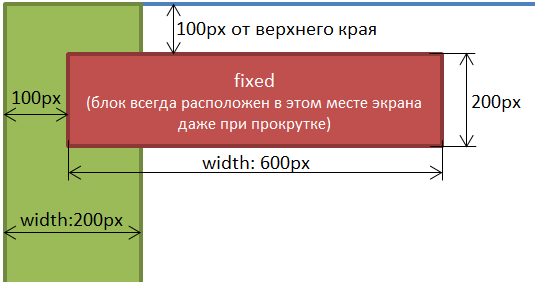
HTMLКопировать

<body>

<div style="width: 200px; height: 2000px; border: 1px solid black; background: green;"></div>

<div style="width: 600px; height: 200px; position: fixed; top: 100px; left: 100px; border: 1px solid black; background:red;"></div>

</body>



## Совместное использование относительного и абсолютного позиционирования

Относительное позиционирование очень часто используется вместе с абсолютным позиционированием.

Рассмотрим варианты:

1. Если расположить блоки с абсолютным позиционированием в блок с относительным, то расстояния будут уже задаваться не от края окна браузера, а от границ относительного блока.

HTMLКопировать

<body>

<!-- Зелёный блок с относительным позиционированием -->

<div style="width: 400px; height:400px; position:relative; top:50px; left:50px; border: 1px solid black; background:green;">

<!-- Красный блок с абсолютным позиционированием -->

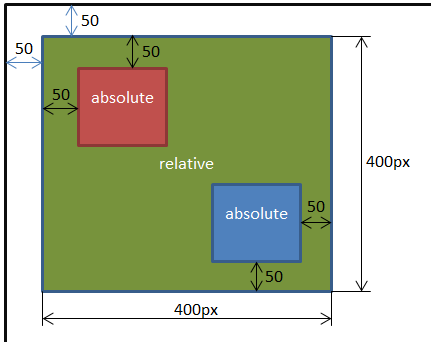
<div style="width: 100px; height:100px; position:absolute; top:50px; left:50px; border: 1px solid black; background:red;"></div>

<!-- Синий блок с абсолютным позиционированием -->

<div style="width: 100px; height:100px; position:absolute; bottom:50px; right:50px; border: 1px solid black; background:blue;"></div>

</div>

</body>



1. Например: для создания фиксированных макетов состоящих из 3 блоков, выровненных по верхнему краю. Установим высоту "400px" относительному блоку для наглядности .

HTMLКопировать

<body>

<div style="width: 800px; height:400px; position:relative; border: 1px solid black; background:green;">

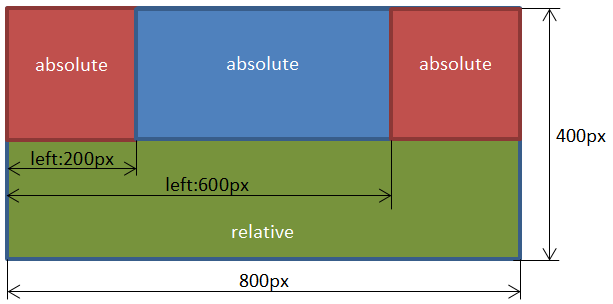
<div style="width: 200px; height:200px; position:absolute; left:0px; border: 1px solid black; background:red;">Левый блок</div>

<div style="width: 400px; height:200px; position:absolute; left:200px; border: 1px solid black; background:blue;">Основной блок</div>

<div style="width: 200px; height:200px; position:absolute; left:600px; border: 1px solid black; background:red;">Правый блок</div>

</div>

</body>



1. Дополнительно к блокам можно применять свойство z-index, которое предназначено для позиционирования элементов по оси Z. Чем больше значение свойства z-index, тем ближе элемент расположен к нам, и наоборот, чем меньше значение, тем дальше расположен элемент от нас.

HTMLКопировать

<body>

<div style="width: 300px; height:300px; position:relative; border: 1px solid black; background:green;">

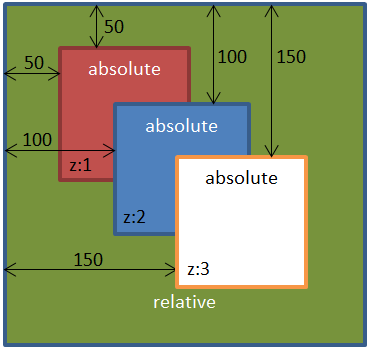
<div style="width: 100px; height:100px; position: absolute; z-index: 1; left: 50px; top: 50px; border: 1px solid black; background: red;"></div>

<div style="width: 100px; height: 100px; position: absolute; z-index: 2; left: 100px; top: 100px; border: 1px solid black; background: blue;"></div>

<div style="width: 100px; height: 100px; position: absolute; z-index: 3; left: 150px; top: 150px; border: 1px solid black; background: yellow;"></div>

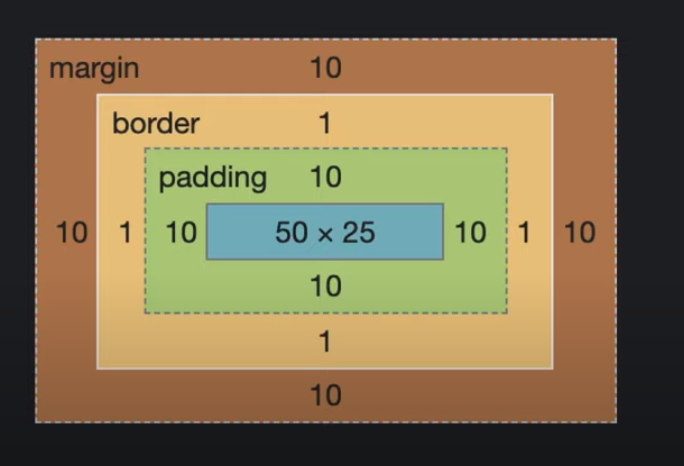
</div>

</body>



<https://www.w3schools.com/css/css_positioning.asp>

[Блочная модель CSS?](https://youtu.be/ycYp7CYOnO0?t=410)

****

https://developer.mozilla.org/ru/docs/Learn/CSS/Building\_blocks/The\_box\_model

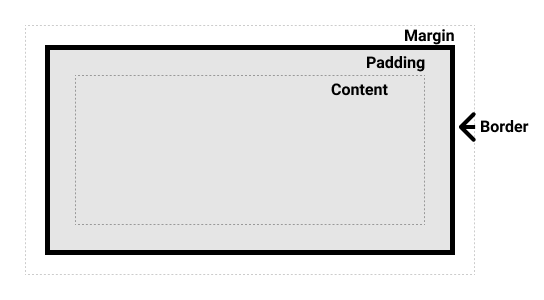
Полностью блочная модель в CSS применяется к блочным элементам, строчные элементы используют не все свойства, определённые блочной моделью. Модель определяет, как разные части элемента — поля, рамки, отступы и содержимое — работают вместе, чтобы создать объект, который вы можете увидеть на странице. Дополнительная сложность заключается в том, что существуют стандартная и альтернативная блочные модели.

### [Составляющие элемента](https://developer.mozilla.org/ru/docs/Learn/CSS/Building_blocks/The_box_model" \l "составляющие_элемента)

Составляя блочный элемент в CSS мы имеем:

* Содержимое: область, где отображается ваш контент, размер которой можно изменить с помощью таких свойств, как [width](https://developer.mozilla.org/ru/docs/Web/CSS/width) и [height](https://developer.mozilla.org/ru/docs/Web/CSS/height).
* Внутренний отступ: отступы располагаются вокруг содержимого в виде пустого пространства; их размер контролируется с помощью [padding](https://developer.mozilla.org/ru/docs/Web/CSS/padding) и связанных свойств.
* Рамка: рамка оборачивает содержимое и внутренние отступы. Её размер и стиль можно контролировать с помощью [border](https://developer.mozilla.org/ru/docs/Web/CSS/border) и связанных свойств.
* Внешний отступ: внешний слой, заключающий в себе содержимое, внутренний отступ и рамки, представляет собой пространство между текущим и другими элементами. Его размер контролируется с помощью [margin](https://developer.mozilla.org/ru/docs/Web/CSS/margin) и связанных свойств.

Рисунок ниже показывает эти слои:



### [Стандартная блочная модель CSS](https://developer.mozilla.org/ru/docs/Learn/CSS/Building_blocks/The_box_model" \l "стандартная_блочная_модель_css)

В стандартной блочной модели, если указать элементу атрибуты width и height, это определит ширину и высоту содержимого. Любые отступы и рамки затем добавляются к этой ширине и высоте для получения общего размера элемента. Это показано на изображении ниже.

Предположим, что в элементе есть следующий CSS определяющий width, height, margin, border, и padding:

.box {

width: 350px;

height: 150px;

margin: 10px;

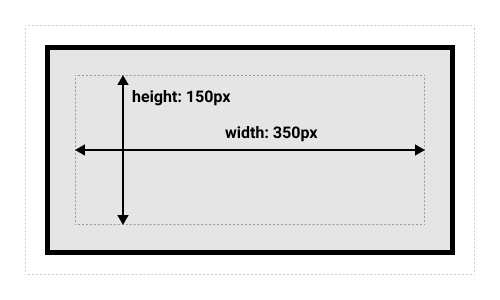
padding: 25px;

border: 5px solid black;

}

Пространство, занимаемое нашим объектом с использованием стандартной блочной модели, на самом деле будет равно 410px в ширину (350 + 25 + 25 + 5 + 5) и 210px в высоту (150 + 25 + 25 + 5 + 5), поскольку отступы и рамки добавляются к размерам поля содержимого.

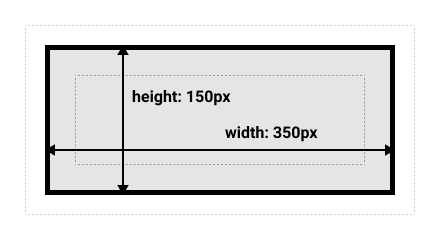
Copy to Clipboard



Примечание: Внешний отступ не считается в фактическом размере объекта. Конечно, он влияет на общее пространство, занимаемое объектом на странице, но только на внешнюю часть. Область элемента заканчивается на рамке — она не распространяется за нее.

### [Альтернативная блочная модель CSS](https://developer.mozilla.org/ru/docs/Learn/CSS/Building_blocks/The_box_model" \l "альтернативная_блочная_модель_css)

Вы можете подумать, что довольно неудобно добавлять рамки и отступы, чтобы получить реальный размер элемента, и окажетесь правы! По этой причине, спустя некоторое время после стандартной блочной модели, в CSS была введена альтернативная блочная модель. При использовании альтернативной модели любая ширина — это ширина видимой части элемента на странице, поэтому ширина области содержимого будет равна общей ширине минус ширина рамки и внутреннего отступа. Тот же CSS, который использовался выше, даст следующий результат (ширина = 350px, высота = 150px).



По умолчанию браузеры используют стандартную блочную модель. Если вы хотите использовать альтернативную блочную модель для элемента, установите для него свойство box-sizing: border-box. С помощью этого вы говорите браузеру о том, что рамка элемента определяется любыми размерами, которые вы устанавливаете.

.box {

box-sizing: border-box;

}

Если вы хотите, чтобы все ваши элементы использовали альтернативную блочную модель, что является распространённым выбором среди разработчиков, установите свойство box-sizing для элемента <html>, затем задайте всем элементам наследование этого значения (inherit), как показано в примере ниже. Если вы хотите понять ход мыслей, стоящий за этим решением, читайте статью [the CSS Tricks article on box-sizing](https://css-tricks.com/inheriting-box-sizing-probably-slightly-better-best-practice/).

Copy to Clipboard

html {

box-sizing: border-box;

}

\*, \*::before, \*::after {

box-sizing: inherit;

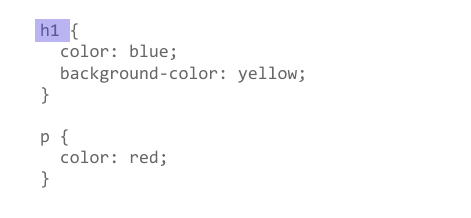
}

Примечание: Интересный факт — Internet Explorer по умолчанию использовал альтернативную блочную модель без доступного механизма для переключения.

Copy to Clipboard

[Что такое селектор? И какие селекторы существуют?](https://youtu.be/G7hLwudGWL4?t=282)

Вы уже встречались с селекторами. Это выражения, которые говорят браузеру, к какому элементу HTML нужно применить те или иные свойства CSS, определённые внутри блока объявления стиля.



Ранее вы встречали несколько разных селекторов и узнали, что существуют селекторы, которые по-разному относятся к документу, — например используя элемент h1 или класс .special.

В CSS селекторы определяются в спецификации CSS-селекторов; как и другие части CSS, нужно поддерживать их работу в браузерах. Большинство селекторов, которые вы встретите, определены в [Спецификации селекторов 3 уровня](https://www.w3.org/TR/selectors-3/), где вы сможете найти всю информацию о поддержке селекторов в браузерах.

## [Несколько селекторов](https://developer.mozilla.org/ru/docs/Learn/CSS/Building_blocks/Selectors" \l "несколько_селекторов)

Несколько селекторов, использующих одни и те же таблицы стилей, можно объединить в лист селекторов: правило будет добавлено к каждому селектору. К примеру, у меня есть одинаковые правила для заголовка h1 и класса .special; я могу написать их так:

h1 {

color: blue;

}

.special {

color: blue;

}

А могу написать короче — просто отделив селекторы запятыми:

Copy to Clipboard

h1, .special {

color: blue;

}

Пробел можно вставлять до или после запятой. Ещё удобнее писать каждый селектор с новой строки:

Copy to Clipboard

h1,

.special {

color: blue;

}

При объединении селекторов таким образом, при условии если хоть один селектор будет недействительным, всё правило будет пропущено.

## [Типы селекторов](https://developer.mozilla.org/ru/docs/Learn/CSS/Building_blocks/Selectors" \l "типы_селекторов)

Понимание того, какой именно селектор вам нужен, очень помогает подобрать подходящий элемент. Сейчас мы разберём разные виды селекторов.

### [Селекторы тегов, классов и идентификаторов](https://developer.mozilla.org/ru/docs/Learn/CSS/Building_blocks/Selectors" \l "селекторы_тегов_классов_и_идентификаторов)

К этой группе относятся селекторы HTML-элементов, таких как <h1>.

h1 { }

К группе относятся и селекторы классов:

Copy to Clipboard

.box { }

или селекторы идентификаторов (ID):

Copy to Clipboard

#unique { }

### Селекторы атрибутов

[Copy to Clipboard](https://developer.mozilla.org/ru/docs/Learn/CSS/Building_blocks/Selectors" \l "селекторы_атрибутов)

Эта группа селекторов позволяет выбирать селекторы, основываясь на наличии у них конкретного атрибута элемента:

a[title] { }

или основываясь на значении атрибута:

Copy to Clipboard

a[href="https://example.com"] { }

### Псевдоклассы, псевдоэлементы

[Copy to Clipboard](https://developer.mozilla.org/ru/docs/Learn/CSS/Building_blocks/Selectors" \l "псевдоклассы_псевдоэлементы)

К этой группе относятся псевдоклассы, которые стилизуют определённое состояние элемента. Псевдокласс :hover, например, применяет правило, только если на элемент наведён курсор мыши

a:hover { }

К группе ещё относятся псевдоэлементы, которые выбирают определённую часть элемента (вместо целого элемента). Например, ::first-line всегда выбирает первую строку внутри элемента (абзаца <p> в нашем случае), действуя, как если бы тег <span> оборачивал первую строку, а затем был стилизован.

Copy to Clipboard

p::first-line { }

### Комбинаторы

[Copy to Clipboard](https://developer.mozilla.org/ru/docs/Learn/CSS/Building_blocks/Selectors" \l "комбинаторы)

И последняя группа селекторов: она позволяет объединять селекторы, чтобы было легче находить конкретные элементы внутри документа. В следующем примере мы отыскали дочерний элемент <article> с помощью комбинатора дочерних элементов (>):

article > p { }

## Продолжение

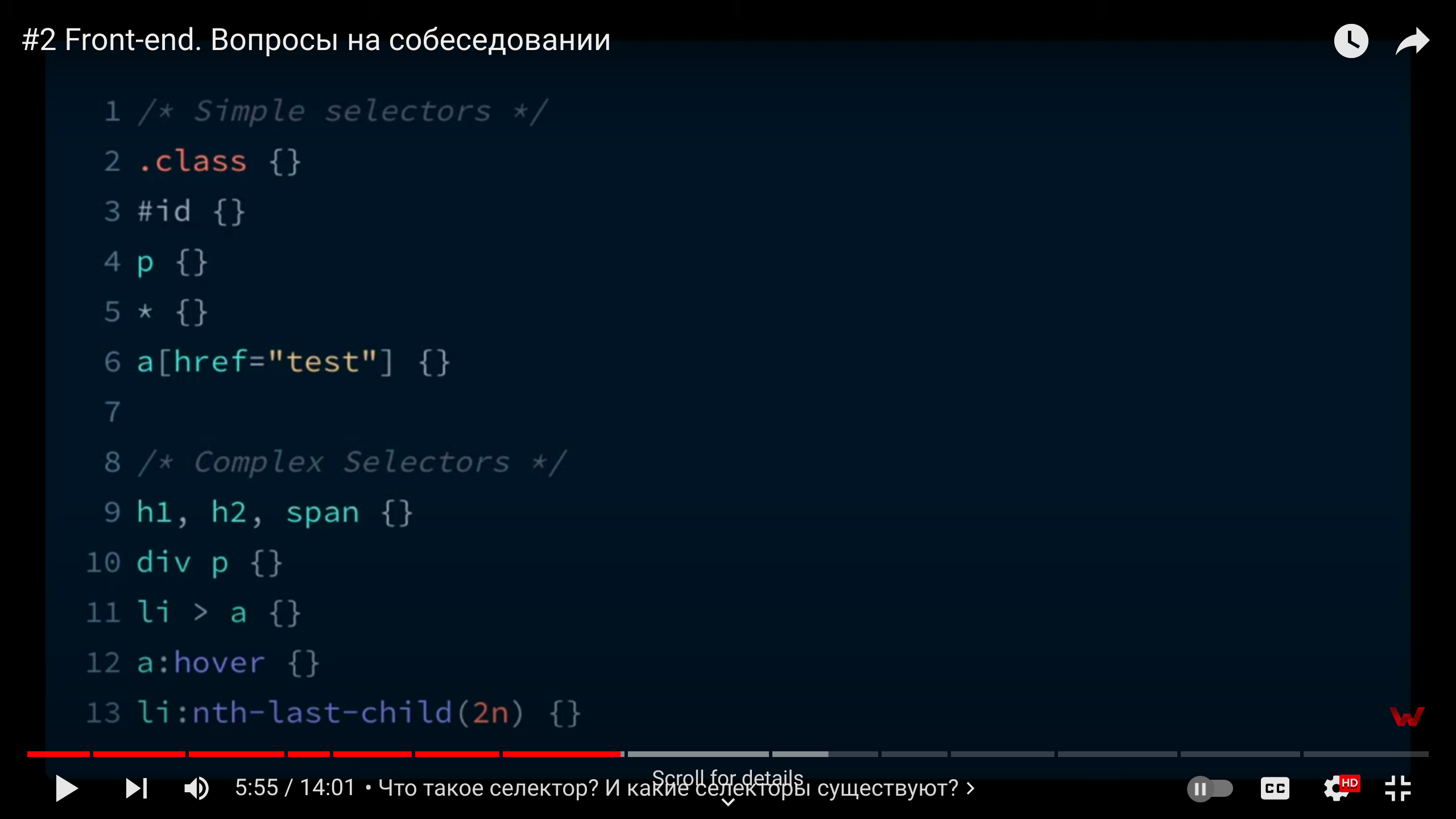
[Copy to Clipboard](https://developer.mozilla.org/ru/docs/Learn/CSS/Building_blocks/Selectors" \l "продолжение)

Ниже можно просмотреть таблицу различных видов селекторов с соответствующими ссылками, или вы можете двинуться дальше: нас ждут [селекторы тегов, классов и идентификаторов.](https://developer.mozilla.org/ru/docs/Learn/CSS/Building_blocks/Selectors/Type_Class_and_ID_Selectors)

## [Справка о селекторах](https://developer.mozilla.org/ru/docs/Learn/CSS/Building_blocks/Selectors" \l "справка_о_селекторах)

В таблице ниже — доступные сейчас селекторы, а также ссылки к страницам, где рассказывается, как использовать каждый из них. Я также добавил ссылки на страницы MDN для каждого селектора, чтобы вы могли проверить, поддерживаются ли они браузерами.

| Селектор | Пример | Руководство |
| --- | --- | --- |
| [Селектор по типу](https://developer.mozilla.org/ru/docs/Web/CSS/Type_selectors) | h1 { } | [Селектор по типу](https://developer.mozilla.org/ru/docs/Learn/CSS/Building_blocks/Selectors/Type_Class_and_ID_Selectors" \l "селекторы_по_типу) |
| [Универсальный селектор](https://developer.mozilla.org/ru/docs/Web/CSS/Universal_selectors) | \* { } | [Универсальный селектор](https://developer.mozilla.org/ru/docs/Learn/CSS/Building_blocks/Selectors/Type_Class_and_ID_Selectors" \l "универсальный_селектор) |
| [Селектор класса](https://developer.mozilla.org/ru/docs/Web/CSS/Class_selectors) | .box { } | [Селекторы классов](https://developer.mozilla.org/ru/docs/Learn/CSS/Building_blocks/Selectors/Type_Class_and_ID_Selectors" \l "селекторы_классов) |
| [Селектор ID](https://developer.mozilla.org/ru/docs/Web/CSS/ID_selectors) | #unique { } | [Селекторы по ID](https://developer.mozilla.org/ru/docs/Learn/CSS/Building_blocks/Selectors/Type_Class_and_ID_Selectors" \l "селекторы_по_id) |
| [Селектор атрибутов](https://developer.mozilla.org/ru/docs/Web/CSS/Attribute_selectors) | a[title] { } | [Селекторы атрибутов](https://developer.mozilla.org/ru/docs/Learn/CSS/Building_blocks/Selectors/Attribute_selectors) |
| [Селектор псевдоклассов](https://developer.mozilla.org/ru/docs/Web/CSS/Pseudo-classes) | p:first-child { } | [Псевдоклассы](https://developer.mozilla.org/ru/docs/Learn/CSS/Building_blocks/Selectors/Pseudo-classes_and_pseudo-elements" \l "что_такое_псевдокласс) |
| [Селектор псевдоэлементов](https://developer.mozilla.org/ru/docs/Web/CSS/Pseudo-elements) | p::first-line { } | [Псевдоэлементы](https://developer.mozilla.org/ru/docs/Learn/CSS/Building_blocks/Selectors/Pseudo-classes_and_pseudo-elements" \l "что_такое_псевдоэлемент) |
| [Селектор потомков](https://developer.mozilla.org/ru/docs/Web/CSS/Descendant_combinator) | article p | [Селектор потомков](https://developer.mozilla.org/ru/docs/Learn/CSS/Building_blocks/Selectors/Combinators" \l "селектор_потомков) |
| [Селектор дочерних элементов](https://developer.mozilla.org/ru/docs/Web/CSS/Child_combinator) | article > p | [Селектор дочерних элементов](https://developer.mozilla.org/ru/docs/Learn/CSS/Building_blocks/Selectors/Combinators" \l "комбинатор) |
| [Смежные селекторы](https://developer.mozilla.org/ru/docs/Web/CSS/Adjacent_sibling_combinator) | h1 + p | [Смежные селекторы](https://developer.mozilla.org/ru/docs/Learn/CSS/Building_blocks/Selectors/Combinators" \l "смежные_селекторы) |
| [Селектор братских элементов](https://developer.mozilla.org/ru/docs/Web/CSS/General_sibling_combinator) | h1 ~ p | [Селектор братских элементов](https://developer.mozilla.org/ru/docs/Learn/CSS/Building_blocks/Selectors/Combinators" \l "братские_элементы) |

Простые: селектор класса, айди, тега/типа элемента, универсальный селектор, селектор атрибута. Составные: групповые селекторы, селектор потомка, селектор дочернего элемента, селектор псевдокласса, селектор псевдоэлемента.

[Что такое специфичность селектора? Как считать вес селектора?](https://youtu.be/G7hLwudGWL4?t=358)

<https://www.w3schools.com/css/css_specificity.asp>

Специфичность - это способ, с помощью которого браузеры определяют, какие значения свойств CSS наиболее соответствуют элементу и, следовательно, будут применены. Специфичность основана на правилах соответствия, состоящих из [селекторов CSS](https://developer.mozilla.org/ru/docs/Web/CSS/Reference" \l "selectors) различных типов.

## [Как вычисляется специфичность?](https://developer.mozilla.org/ru/docs/Web/CSS/Specificity" \l "как_вычисляется_специфичность)

Специфичность представляет собой вес, придаваемый конкретному правилу CSS. Вес правила определяется количеством каждого из [типов селекторов](https://developer.mozilla.org/ru/docs/Web/CSS/Specificity" \l "selector_types) в данном правиле. Если у нескольких правил специфичность одинакова, то к элементу применяется последнее по порядку правило CSS. Специфичность имеет значение только в том случае, если один элемент соответствует нескольким правилам. Согласно спецификации CSS, правило для непосредственно соответствующего элемента всегда будет иметь больший приоритет, чем правила, унаследованные от предка.

Примечание: Примечание: Взаимное расположение элементов в дереве документа не влияет на специфичность.

### [Типы селекторов](https://developer.mozilla.org/ru/docs/Web/CSS/Specificity" \l "типы_селекторов)

В следующем списке типы селекторов расположены по возрастанию специфичности:

1. селекторы типов элементов (например, h1) и псевдоэлементов (например, ::before).
2. селекторы классов (например, .example), селекторы атрибутов (например, [type="radio"]) и псевдоклассов (например, :hover).
3. селекторы идентификаторов (например, #example).

Универсальный селектор (\*), комбинаторы (+, >, ~, '``') и отрицающий псевдокласс (:not()) не влияют на специфичность. (Однако селекторы, объявленные внутри :not(), влияют)

Стили, объявленные в элементе (например, style="font-weight:bold"), всегда переопределяют любые правила из внешних файлов стилей и, таким образом, их специфичность можно считать наивысшей.

### [Важное исключение из правил -](https://developer.mozilla.org/ru/docs/Web/CSS/Specificity" \l "важное_исключение_из_правил_-_!important)!important

Когда при объявлении стиля используется модификатор !important, это объявление получает наивысший приоритет среди всех прочих объявлений. Хотя технически модификатор !important не имеет со специфичностью ничего общего, он непосредственно на неё влияет. Поскольку !important усложняет отладку, нарушая естественное [каскадирование](https://developer.mozilla.org/ru/docs/Web/CSS/Cascade) ваших стилей, он не приветствуется и следует избегать его использования. Если к элементу применимы два взаимоисключающих стиля с модификатором !important, то применён будет стиль с большей специфичностью.

Несколько практических советов:

* Всегда пытайтесь использовать специфичность, а !important используйте только в крайних случаях
* Используйте !important только в страничных стилях, которые переопределяют стили сайта или внешние стили (стили библиотек, таких как Bootstrap или normalize.css)
* Никогда не используйте !important, если вы пишете плагин или мэшап.
* Никогда не используйте !important в общем CSS сайта.

Вместо !important можно:

1. Лучше использовать каскадные свойства CSS
2. Использовать более специфичные правила. Чтобы сделать правило более специфичным и повысить его приоритет, укажите один элемент или несколько перед нужным вам элементом:

<div id="test">

<span>Text</span>

</div>

div#test span { color: green }

Copy to Clipboard

div span { color: blue }

span { color: red }

Copy to Clipboard

Вне зависимости от порядка следования правил, текст всегда будет зелёным, поскольку у этого правила наибольшая специфичность (при этом, правило для голубого цвета имеет преимущество перед правилом для красного, несмотря на порядок следования).

Вам придётся использовать !important если:

А) Первый сценарий:

1. У вас есть общий файл стилей, устанавливающий правила для внешнего вида сайта.
2. Вы пользуетесь (или кто-то другой пользуется) весьма сомнительным средством - объявлением стилей непосредственно в элементах

В таком случае вам придётся объявить некоторые стили в вашем общем файле CSS как !important, переопределяя, таким образом, стили, установленные в самих элементах.

Пример из практики: Некоторые плохо написанные плагины jQuery, использующие присваивание стилей самим элементам.

Б) Ещё сценарий:

#someElement p {

color: blue;

}

p.awesome {

color: red;

}

Как сделать цвет текста в абзацах awesome красным всегда, даже если они расположены внутри #someElement? Без !important у первого правила специфичность больше и оно имеет преимущество перед вторым.

Copy to Clipboard

Как преодолеть !important

A) Просто добавьте ещё одно правило с модификатором !important, у которого селектор имеет большую специфичность (благодаря добавлению типа элемента (тэга), идентификатора (атрибута id) или класса к селектору).

Пример большей специфичности:

table td {height: 50px **!important**;}

.myTable td {height: 50px **!important**;}

#myTable td {height: 50px **!important**;}

Б) Или добавьте правило с модификатором !important и таким же селектором, но расположенное в файле после существующего (при прочих равных выигрывает последнее объявленное правило):

Copy to Clipboard

td {height: 50px **!important**;}

В) Или перепишите первоначальное правило без использования !important.

Copy to Clipboard

С более подробной информацией можно ознакомиться по следующим ссылкам:

[Когда надо использовать !important в CSS?](https://stackoverflow.com/questions/3706819/what-are-the-implications-of-using-important-in-css)

[Что означает !important в CSS?](https://stackoverflow.com/questions/9245353/what-does-important-in-css-mean)

[Когда в CSS надо использовать модификатор !important](https://stackoverflow.com/questions/5701149/when-to-use-important-property-in-css)

[Как преодолеть !important](https://stackoverflow.com/questions/11178673/how-to-override-important)

[Как использовать модификатор !important в CSS чтобы сэкономить время](https://stackoverflow.com/questions/2042497/when-to-use-important-to-save-the-day-when-working-with-css)

### [Не исключение -](https://developer.mozilla.org/ru/docs/Web/CSS/Specificity" \l "не_исключение_-_not):not()

Отрицающий псевдокласс :not не учитывается как псевдокласс при расчёте специфичности. Однако селекторы, расположенные внутри :not, при подсчёте количества [по типам селекторов](https://developer.mozilla.org/ru/docs/Web/CSS/Specificity" \l "selector-type) рассматриваются как обычные селекторы и учитываются.

Следующий фрагмент CSS ...

div.outer p {

color: orange;

}

div:not(.outer) p {

color: lime;

}

... применённый к такому HTML ...

Copy to Clipboard

<div class="outer">

<p>Это div.outer</p>

<div class="inner">

<p>Это текст в div.inner</p>

</div>

</div>

... отобразится на экране так:

Copy to Clipboard

Это div.outer

Это текст в div.inner

### [Специфичность основана на форме](https://developer.mozilla.org/ru/docs/Web/CSS/Specificity" \l "специфичность_основана_на_форме)

Специфичность опирается на форму селектора. В следующем примере, при определении специфичности селектора, селектор \*[id="foo"] считается селектором атрибута, даже при том, что ищет идентификатор.

Эти объявления стилей ...

\*#foo {

color: green;

}

\*[id="foo"] {

color: purple;

}

... применённые к нижеследующей разметке ...

Copy to Clipboard

<p id="foo">Это пример.</p>

... в результате выглядят так:

Copy to Clipboard

Это пример.

Потому что оба правила соответствуют одному и тому же элементу, но селектор идентификатора имеет большую специфичность.

### [Независимость от расположения](https://developer.mozilla.org/ru/docs/Web/CSS/Specificity" \l "независимость_от_расположения)

Взаимное расположение элементов, указанных в селекторе не влияет на специфичность правила. Следующие объявления стилей ...

body h1 {

color: green;

}

html h1 {

color: purple;

}

... в сочетании со следующим HTML ...

Copy to Clipboard

<html>

<body>

<h1>Вот заголовок!</h1>

</body>

</html>

... отобразится как:

Copy to Clipboard

Вот заголовок!

Потому что, хотя оба объявления имеют одинаковое количество [типов селекторов](https://developer.mozilla.org/ru/docs/Web/CSS/Specificity" \l "selector-type), но селектор html h1 объявлен последним.

### [Непосредственно соответствующие элементы и унаследованные стили](https://developer.mozilla.org/ru/docs/Web/CSS/Specificity" \l "непосредственно_соответствующие_элементы_и_унаследованные_стили)

Стили непосредственно соответствующих элементов всегда предпочитаются унаследованным стилям, независимо от специфичности унаследованного правила. Этот CSS ...

#parent {

color: green;

}

h1 {

color: purple;

}

... с таким HTML ...

Copy to Clipboard

<html>

<body id="parent">

<h1>Вот заголовок!</h1>

</body>

</html>

... тоже отобразится как:

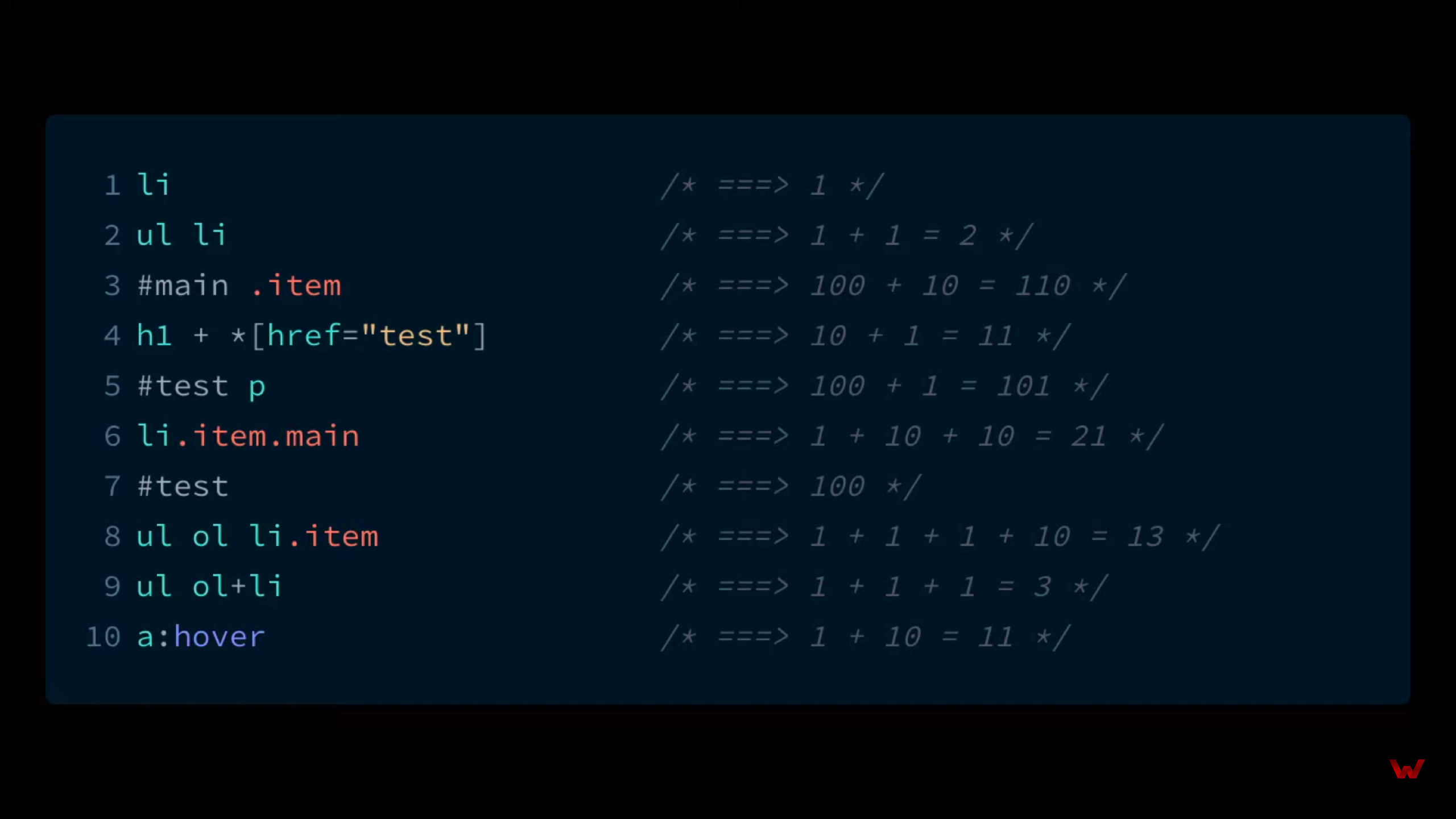
Copy to Clipboard

Вот заголовок!

Потому что селектор h1 непосредственно соответствует элементу, а стиль, задающий зелёный цвет, всего лишь унаследован от родителя.

## [Дополнительная информация](https://developer.mozilla.org/ru/docs/Web/CSS/Specificity" \l "дополнительная_информация)

* Калькулятор специфичности: Интерактивный сайт, помогающий вам проверить и понять ваши собственные правила CSS - <https://specificity.keegan.st/>
* Специфичность селекторов в CSS3 - [http://www.w3.org/TR/selectors/#specificity](_blank)

****

****

[Разница между Reset.css и Normalize.css?](https://youtu.be/G7hLwudGWL4?t=445)

Каждый элемент обладает дефолтными стилями, у каждого браузера свои особенности. Для кроссбраузерности нужно применить либо рисет, либо нормалайз.

Если вы когда-нибудь делали коммерческий заказ на верстку, то наверняка сталкивались с таким требованием как кроссбраузерность, то есть чтобы ваша страница отображалась максимально одинаково во всех браузерах. И поэтому возникает вопрос – что же выбрать, reset или normalize?

## **Что лучше reset.css или normalize.css?**

Для начала давайте выясним что делает каждый из файлов:

**reset.css** – как и видно по названию файла, сбрасывает большинство стилей браузера, которые в нем заданы по умолчанию (а такие действительно есть, просто откройте голый HTML).

**normalize.css** – этот файл не только задает стили по умолчанию там где это необходимо, но и исправляет некоторые недостатки старых браузеров (привет IE). Сохраняет дефолтные стили, делает их одинаковыми.

Как вы понимаете, эти 2 файла даже не совсем корректно сравнивать, так как у них немного разная “философия”. Так в чем же разница и что выбрать?

## **В чем отличия normalize от reset css**

1. В отличие от reset, после подключения normalize css, вы визуально сможете определить где у вас находятся различные элементы, те же параграфы имеют внешние отступы. Поэтому, применяя “таблицу стилей со сбросом”, вы просто приведете большое количество элементов к одному виду.
2. Исходя из 1 правила следует что нам нужно написать стили с нуля, т.к. у нас всё сброшено, что лично по мне, уже не комильфо.
3. В normalize css есть исправления различных общих ошибок, таких как отображение HTML5 тегов или тех же форм
4. Когда вы работаете с инструментами отладки в браузерах, таких, например, как фаербаг, подключая ресет вы получите огромную “простыню” из свойств, которая в итоге увеличит ваше время на написание новых стилей и поиск старых
5. В normalize каждое отдельное правило задокументировано и вы легко сможете понять для чего оно там, и если вы уверены что оно вам не нужно, то сможете легко его удалить
6. normalize.css подключен по умолчанию в bootstrap 

Думаю вы и сами понимаете, что **normalize** выигрывает по всем параметрам. Стоит также вспомнить про извращенцев, которые используют что-то в этом стиле:

\* {margin: 0; padding: 0;}

Как вы понимаете так делать не стоит.

В заключении хотелось бы сказать, что я еще не встречал человека, который бы утверждал что reset.css вообще есть смысл применять. Мне кажется, его применяли только потому что не знали про “нормализацию стилей” 

Есть мнение, что стоит подключать стили в head “инлайново”, хотя бы для “первого экрана” – то есть то, что сразу покажет человеку когда он откроет ваш сайт, и это действительно очень хорошая практика. Данную рекомендацию вы можете встретить в [google page speed](https://developers.google.com/speed/pagespeed/insights/).

Возможно вам пригодится – [inline normalize css and bootstrap grid](https://forwww.com/files/inline_style_normalize_and_bootstrap_grid.css)  Пишем это в head, затем также само пишем свои стили для первого экрана, и вы выполните одну из рекомендаций + ваш сайт не будет шататься при загрузке, т.к. основные самые важные стили, а именно сетка, нормалайз и “первый экран” уже прогрузились.

[Разница между margin и padding?](https://youtu.be/1eIRTdgzHtw?t=116)

Падинги (свойство padding ) — это внутренние отступы от границ элемента до его содержимого. Маржины (свойство margin ) — это внешние отступы от границ элемента до соседних элементов. См. Блочную модель.

[Разница между display: none и visibility: hidden?](https://youtu.be/1eIRTdgzHtw?t=139)

Оба правила предназначены для того, чтобы скрывать элемент на странице. Но механизмы, которые они используют, отличаются. Display: none полностью убирает элемент со страницы. Элемент действительно удаляется с основного потока документа. Он остается доступным только в дом-дереве. Контент, находящийся внутри него, становится недоступным для поисковых роботов.

Что касается visibility: hidden, то несмотря на то, что элемент не виден на странице, из основного потока документа он не вырывается и по-прежнему занимает отведенное для него место, а также остается доступным для поисковых роботов.

[О базовом потоке](https://developer.mozilla.org/ru/docs/Learn/CSS/CSS_layout/Normal_Flow)

[Еще о потоке документа](https://doka.guide/html/flow/)

[Разница между блочным и строчным (инлайновым) элементами?](https://youtu.be/1eIRTdgzHtw?t=188)

## Блочные элементы

Блочные элементы — отдельные структурные единицы, которые используются для создания структуры веб-страницы. Такие блоки имеют форму прямоугольника и занимают всю ширину экрана.  Размеры блочных элементов могут быть изменены  с помощью значений свойств *width* и *height*. При помощи свойства *border* можно задавать толщину, цвет и стиль линий границы, по умолчанию она невидима.

По умолчанию границы вплотную примыкают к контенту блоков, увеличить внутренний отступ  можно с помощью  значений свойства *padding*.

При необходимости можно использовать свойство *box-sizing*, которое позволяет изменить алгоритм расчета ширины и высоты, либо учитывая *padding* и *border* при расчете, либо нет, в зависимости от значения свойства *box-sizing*.

## **Строчные элементы**

Стили, предназначенные для строчных элементов не работают на блочных (например,  *vertical-align*).

Строчным элементам установлено по умолчанию значение *dipslay: inline* и они являются частью строки.

Ширина определяется  шириной контента  и  значениями *padding* и *border* (если они есть).

Элемент можно выравнивать по вертикали относительно родительского блока с помощью свойства *vertical-align*.

Еще одно отличие заключается в разных принципах расположения контента: в блочной модели — он вертикальный, а в строчной — горизонтальный.

Что касается совместного использования, то внутри блочных элементов можно размещать элементы обоих типов, внутри строчных — только строчные.

## The display: inline-block Value

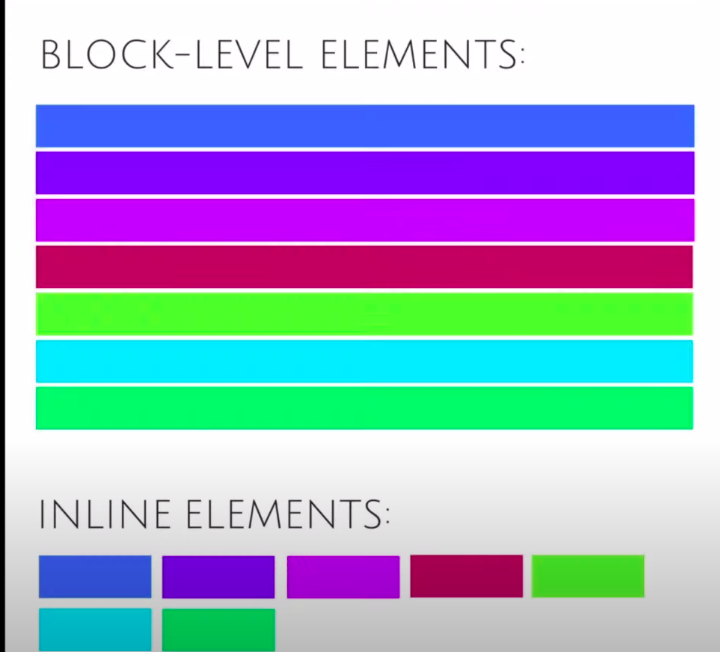
Compared to display: inline, the major difference is that display: inline-block allows to set a width and height on the element.

Also, with display: inline-block, the top and bottom margins/paddings are respected, but with display: inline they are not.

Compared to display: block, the major difference is that display: inline-block does not add a line-break after the element, so the element can sit next to other elements.

## Using inline-block to Create Navigation Links

One common use for display: inline-block is to display list items horizontally instead of vertically. The following example creates horizontal navigation links:

****

[Разница между классом и идентификатором в CSS?](https://youtu.be/CjdCxxqObaM?t=264)

****

Разница между идентификатором и классом состоит в том, что идентификатор может присваиваться только одному элементу, а класс – скольким угодно элементам.

Также можно определять селектор CSS для конкретного элемента HTML поместив перед именем класса HTML селектор, так **p.jam { что-нибудь }** определит стиль для элементов параграфов, у которых класс "jam".

[Что такое CSS спрайт? И для чего он используется?](https://youtu.be/CjdCxxqObaM?t=330)

CSS-спрайты — технология, которую разработчики применяют для оптимизации веб-страниц. Технология предполагает объединение нескольких небольших изображений с одинаковым разрешением в одно большое изображение, которое также называется таблицей спрайтов.

аблица спрайтов используется, когда нужно показывать пользователям отдельные элементы, которые являются частью большого изображения.

Обычно веб-разработчики включают в таблицу спрайтов логотипы, навигационные стрелки, кнопки и так далее. Такие элементы имеют одинаковое разрешение и часто используются на страницах сайта.

### Как спрайты помогают веб-разработчикам

Чаще всего при создании сайтов разработчики хранят и используют изображения как отдельные файлы. Когда пользователь открывает страницу, браузер отправляет отдельный [HTTP-запрос](https://ru.hexlet.io/courses/http_protocol), чтобы загрузить и отобразить каждый файл. Это увеличивает время загрузки сайта, так как на странице может быть много отдельных изображений, например, кнопок, иконок, логотипов.

CSS-спрайты позволяют разработчикам объединять часто используемые маленькие изображения в одно большое. Благодаря этому браузер загружает один файл. Чтобы отобразить нужное маленькое изображение, используется смещение в большом изображении.

## Преимущества использования спрайтов

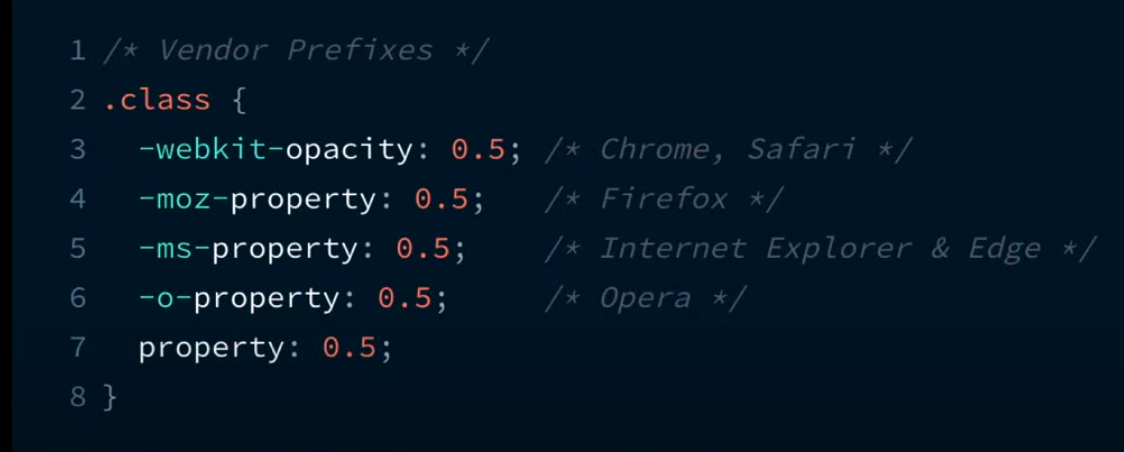
Два основных преимущества использования спрайтов:

1. Повышение скорости загрузки страниц. Изображения на странице становятся доступными, как только браузер загружает один файл: таблицу спрайтов.
2. Увеличение пропускной способности и снижение потребления ресурсов. CSS-спрайты не только улучшают опыт конечного пользователя сайта за счёт более быстрой загрузки страниц. Это технология [уменьшает нагрузку на сеть](https://ru.hexlet.io/blog/posts/http-3-proshloe-nastoyaschee-i-buduschee) за счёт снижения количества HTTP-запросов.

[Подробнее](https://ru.hexlet.io/blog/posts/kak-ispolzovat-css-sprayty-dlya-uvelicheniya-skorosti-zagruzki-veb-stranits" \l ":~:text=CSS-спрайты — технология%2C которую,которое также называется таблицей спрайтов.)

[Что такое вендорные префиксы? И для чего они используются?](https://youtu.be/rlWgI7AvV18?t=234)

Вендерный префикс – это приставка к CSS свойству, который обеспечивает поддержку данного свойства браузерами, в которых оно не внедрено на постоянной основе: то есть, свойство введено в спецификацию CSS, но в конкретном браузере она находится либо в стадии разработки, либо в стадии тестирования. Причин для их появления несколько: включение в браузер экспериментальных свойств CSS, которые стандартом еще не утверждены, а также решение проблем с кроссбраузерностью. При помощи ресурса Can I use можно узнать, как то или иное свойство CSS поддерживается браузерами.

****

Большинство производителей называют несколько причин, когда нужно использовать вендорные префиксы. Основные, из которых:

* Свойство, которое было написано только для конкретного браузера и не содержится в стандартном списке css.
* Свойство ещё разрабатывается или по каким-то причинам не имеет рекомендаций к использованию
* Css задаёт только часть возможностей стиля.

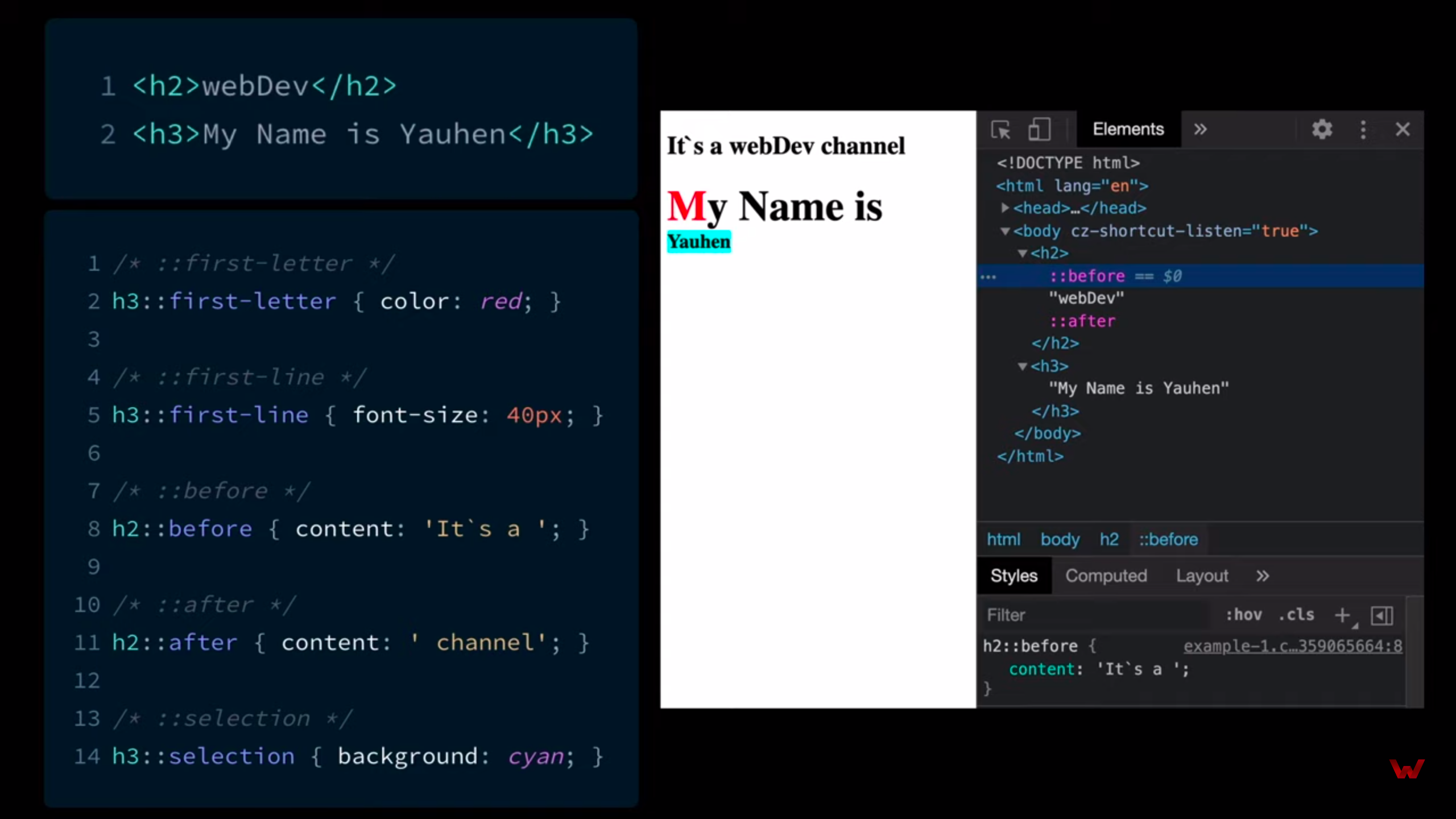
Разработчики компании Microsoft, помимо выше перечисленных причин, при помощи своего вендорного префикса -ms-, прячут от валидатора те конструкции, которые его не пройдут. При верстке не возникает ни какого желания прописывать свойство заново и добавлять ему вендорный префикс, однако эта необходимость вызвана тем, что CSS и его спецификация не всегда содержит необходимые нам свойства. Кроме того, есть не маленький шанс, что описание свойства в дальнейшем измениться и использование префиксов будет не нужным.

[Еще инфо](https://doka.guide/css/vendor-prefixes/)

[Что такое псевдоэлементы? И для чего они используются?](https://youtu.be/rlWgI7AvV18?t=355)

Псевдоэлемент – это ключевое слово, которое доабвляется на селектор и позволяет стилизировать определенную часть выбранного элемента.

5 типов:

[https://www.w3schools.com/css/css\_pseudo\_elements.asp](https://www.w3schools.com/css/css_pseudo_elements.asp)

[Что такое схлопывание границ (margin collapsing)?](https://youtu.be/kx3dR6ztICU?t=30)

## Margin Collapse

Top and bottom margins of elements are sometimes collapsed into a single margin that is equal to the largest of the two margins.

This does not happen on left and right margins! Only top and bottom margins!

Look at the following example:

### Example

h1 {  
  margin: 0 0 50px 0;  
}  
  
h2 {  
  margin: 20px 0 0 0;  
}

In the example above, the <h1> element has a bottom margin of 50px and the <h2> element has a top margin set to 20px.

Common sense would seem to suggest that the vertical margin between the <h1> and the <h2> would be a total of 70px (50px + 20px). But due to margin collapse, the actual margin ends up being 50px.

Отступы [margin-top](https://developer.mozilla.org/ru/docs/Web/CSS/margin-top) и [margin-bottom](https://developer.mozilla.org/ru/docs/Web/CSS/margin-bottom) иногда объединяются в один, с размером равным наибольшему из них (или размеру одного, если они равны). Это поведение известно как схлопывание внешних отступов (margin collapsing). Обратите внимание, что отступы [плавающих](https://developer.mozilla.org/ru/docs/Web/CSS/float) и [абсолютно позиционированных](https://developer.mozilla.org/ru/docs/Web/CSS/position" \l "absolute) элементов никогда не схлопываются.

Схлопывание внешних отступов происходит в трёх случаях:

Соседние элементы (siblings)

Схлопываются отступы соседних элементов (за исключением случая, когда к последнему элементу применено свойство [clear](https://developer.mozilla.org/ru/docs/Web/CSS/clear)).

Родительский и первый/последний дочерние элементы

Если отсутствуют границы (border), внутренние отступы (padding), строчное содержимое (inline/inline-block) или промежуток для отделения [margin-top](https://developer.mozilla.org/ru/docs/Web/CSS/margin-top) родительского элемента, от [margin-top](https://developer.mozilla.org/ru/docs/Web/CSS/margin-top) одного или нескольких его дочерних элементов/блоков или отсутствуют границы (border), внутренние отступы (padding), строчное содержимое (inline/inline-block), [height](https://developer.mozilla.org/ru/docs/Web/CSS/height), [min-height](https://developer.mozilla.org/ru/docs/Web/CSS/min-height) или [max-height](https://developer.mozilla.org/ru/docs/Web/CSS/max-height) для отделения отступов [margin-bottom](https://developer.mozilla.org/ru/docs/Web/CSS/margin-bottom) родительского блока от [margin-bottom](https://developer.mozilla.org/ru/docs/Web/CSS/margin-bottom) отступов одного или нескольких его дочерних элементов/блоков, то внешние отступы схлопываются. Схлопнутые отступы заканчиваются за пределами родительского элемента.

Пустые блоки

Если отсутствуют границы (border), внутренние отступы (padding), строчное содержимое (inline/inline-block), [height](https://developer.mozilla.org/ru/docs/Web/CSS/height) или [min-height](https://developer.mozilla.org/ru/docs/Web/CSS/min-height) для отделения [margin-top](https://developer.mozilla.org/ru/docs/Web/CSS/margin-top) верхнего отступа этого блока от его [margin-bottom](https://developer.mozilla.org/ru/docs/Web/CSS/margin-bottom) нижнего отступа, то верхние и нижние внешние отступы этого блока схлопываются.

На заметку:

* Более сложное схлопывание отступов (более, чем двух) происходит, когда описанные случаи сочетаются.
* Эти правила применяются даже к отступам, равным 0, поэтому отступ первого/последнего дочернего элемента заканчивается за пределами его родителя (согласно правилу выше) независимо от того, равен ли отступ родителя нулю.
* При использовании отрицательных отступов, размер схлопнутого отступа вычисляется, как сумма наибольшего положительного и наименьшего отрицательного (наибольшего по модулю) отступа.
* Если все отступы отрицательные, размер схлопнутого отступа равен наименьшему (наибольшему по модулю) отступу. Это относится как к вложенным элементам, так и к соседним.
* Внешние отступы [плавающих](https://developer.mozilla.org/ru/docs/Web/CSS/float) и [абсолютно позиционируемых](https://developer.mozilla.org/ru/docs/Web/CSS/position) элементов никогда не схлопываются.

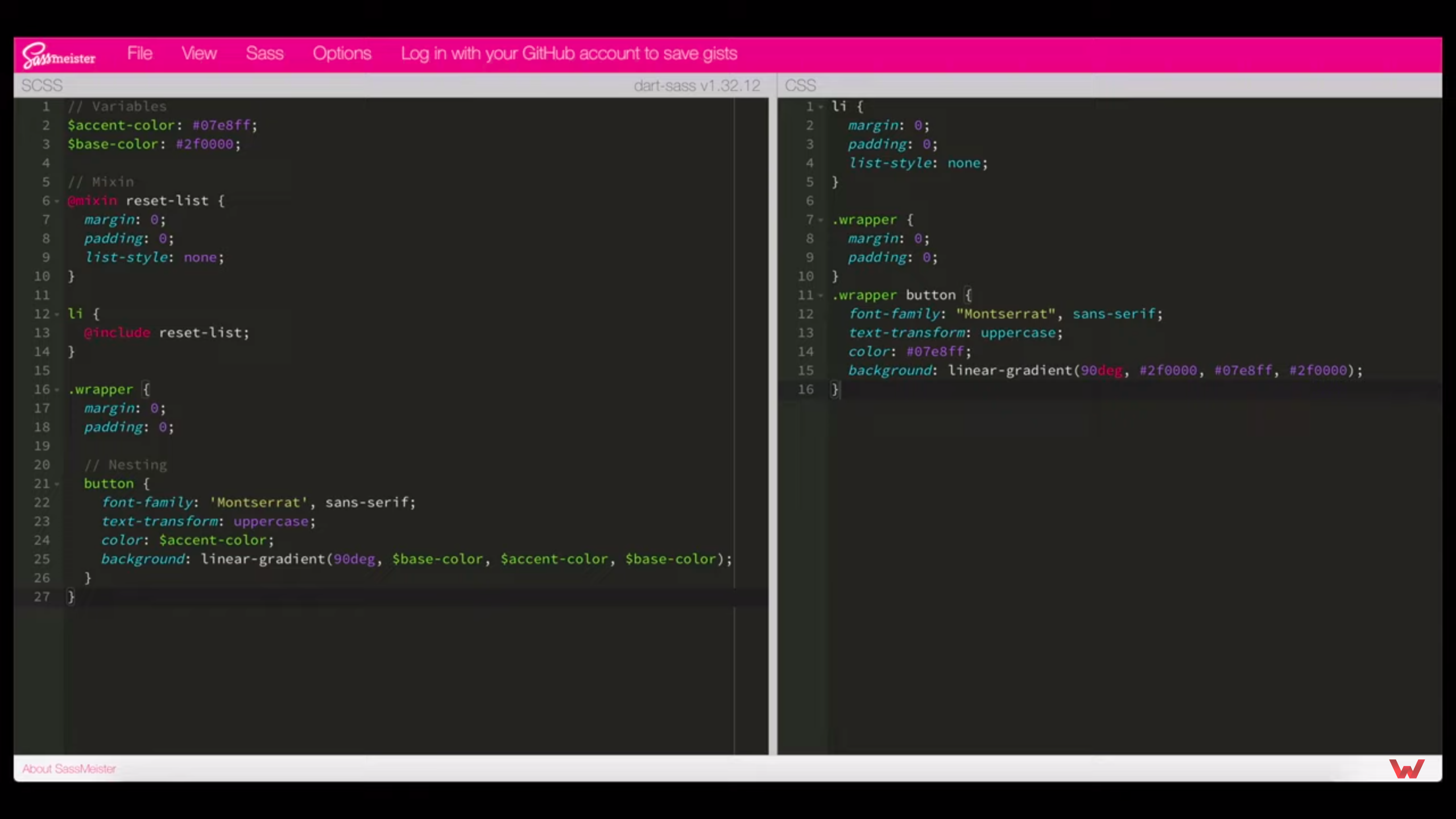
[Что такое CSS препроцессор?](https://youtu.be/kx3dR6ztICU?t=159)

https://liquidhub.ru/blogs/blog/preprotsessory-v-css

CSS препроцессор – это программа, которая позволяет генерирвоать CSS из собственного уникального синтаксиса. То есть, на выходе мы всегда будем получать сгенерированный CSS код, который отправляется в браузер. Ну а сами препроцессоры очень активно применяются на разных проектах, поскольку добавляют синтаксис, который зачастую проще читать, поддерживать, переиспользовать и расширять. В CSS препроцессорах есть фичи такие, как:

* переменные (в нативном CSS они тоже появились, но пришли намного позже),
* нестинг или вложенность, когда один селектор можно вкладывать в другой,
* комбинированный селектор вложенности, когда итоговое имя селектора генерируется на освновании вложенности,
* миксины или примеси – переиспользуемые куски кода и т.д.

Самые популярные сейчас – SaaS, LESS, Stylus.

[Что такое z-index? Как формируется контекст наложения?](https://youtu.be/GZUy2i6QN7o?t=102)

Свойство z-index определяет порядок наложения элементов по оси Z - то есть ближе или дальше от взгляда пользователя.

Свойство z-index принимает целочисленные значения. Чем значение меньше - тем элемент дальше, чем значение больше - тем элемент ближе. Также, свойство z-index может принимать отрицательные значения. По-умолчанию имеет значение auto

Свойство z-index работает только с **позиционированными** элементами. По-умолчанию, у всех элементов, свойство position имеет значение static - такой элемент является **не позиционированным**. Чтобы свойство z-index работало, элементу необходимо задать свойство position с одним из следующих значений: relative, absolute, sticky, fixed

Для полного понимания, как работает свойство z-index, рассмотрим такую концепцию, как **контекст наложения**

## Контекст наложения (stacking context)

Для простоты понимания, представим, что каждый элемент - это абстрактная коробка. Корневой элемент <html> - это основная, самая большая коробка, в которую будут складываться другие коробки поменьше. Коробки могут быть открытыми и закрытыми. Если коробка закрыта, то внутренние коробки не могут выйти за пределы этой коробки. Закрытая коробка - это и есть **контекст наложения**.

Другими словами, если у родительского элемента сформирован **контекст наложения**, то свойство z-index у его дочерних элементов, будет определять порядок их наложения только в пределах родительского элемента

У элемента формируется **контекст наложения** (то есть коробка закрывается), если выполняется одно из следующих условий:

1. Элемент является корневым (<html>)
2. Элемент имеет свойство position со значением relative или absolute и свойством z-index со значением, отличным от auto - то есть, с целочисленным значением
3. Элемент имеет свойство position со значением sticky или fixed
4. Элемент является дочерним элементом для родительского **flexbox** элемента и свойством z-index со значением, отличным от auto
5. Элемент является дочерним элементом для родительского **grid** элемента и свойством z-index со значением, отличным от auto
6. Элемент имеет свойство opacity меньше единицы
7. Элемент имеет свойство mix-blend-mode со значением, отличным от normal
8. Элемент имеет одно из следующих свойств со значением, отличным от none: transform, filter, perspective, clip-path, mask/mask-image/mask-border
9. Элемент имеет свойство isolation со значением isolate

Есть еще несколько специфичных свойств, при которых также формируется контекст наложения, с ними можно познакомится на [MDN Web Docs](https://developer.mozilla.org/en-US/docs/Web/CSS/CSS_Positioning/Understanding_z_index/The_stacking_context)

[More info](https://frontips.ru/svojstvo-z-index-i-kontekst-nalozheniya/)

https://developer.mozilla.org/ru/docs/Web/CSS/CSS\_Positioning/Understanding\_z\_index/The\_stacking\_context

Z-index: наслаивание элементов

Позиционированные элементы можно наслаивать друг на друга при помощи CSS-свойства **z-index**, тем самым имитируя третье измерение (ось Z, которая перпендикулярна экрану). Эффект можно сравнить со слоями в программе Adobe Photoshop — вы можете определять, какой элемент будет располагаться выше, а какой — ниже. Давайте подробнее разберем принцип работы z-index.

## Как работает свойство z-index

Свойство **z-index** имеет влияние лишь на позиционированные элементы, т. е. такие, для которых установлено свойство **position** со значением **absolute**, **relative** либо **fixed**. Влияние свойства z-index распространяется как на сам элемент, так и на его дочерние элементы, т. е. поднимая родительский элемент выше по оси Z, вы поднимаете и его дочерние элементы.

Если свойство **z-index** не задано, то позиционированные элементы накладываются друг на друга в том порядке, в котором они находятся в коде HTML. Это значит, что если на странице есть три абсолютно позиционированных тега, то второй тег из HTML отобразится выше первого, а третий — выше первого и второго.

Ниже показано три примера: на первом скриншоте имеется три абсолютно позиционированных элемента **<span>** без указания **z-index**. Как следствие, элементы накладываются друг на друга в обычном порядке. На втором скриншоте добавлено свойство **z-index: 1** к первому элементу **<span>**. Как видим, порядок наложения изменился: теперь первый элемент расположен выше остальных. На третьем скриншоте показано, как можно добиться того же результата, но с применением **z-index: 1** к родителю первого тега **<span>** — тегу **<div>**:

Next

**1**23

## Значения z-index

Свойство **z-index** принимает числовые значения — подходят целые числа (положительные, отрицательные, ноль). Чем больше число, тем выше находится элемент по оси Z. При равных значениях z-index элементы накладываются в порядке следования в HTML.

Значением свойства z-index по умолчанию является **auto**.

## Z-index и контекст наложения

Вышеописанная способность элементов изменять порядок наложения друг на друга с помощью свойства **z-index** работает лишь в том случае, если эти элементы существуют в одном контексте наложения. Что это значит?

**Контекст наложения** (англ. *stacking context*) — это концепция трехмерного размещения HTML-элементов по оси Z, расположенной перпендикулярно экрану. Контекст наложения может быть сформирован любым элементом, который соответствует хотя бы одному из следующих условий:

* Элемент является корневым, т. е. существует в корневом контексте наложения. Любой элемент веб-страницы является таковым, если только он не присутствует в локальном контексте наложения (в том, который создается любым из способов ниже).
* Абсолютно позиционированный (*position: absolute*) либо относительно позиционированный (*position: relative*) элемент с любым значением **z-index**, кроме **auto**.
* Элемент со свойством **position: fixed** и любым значением **z-index**.

Элемент со свойством **display: flex** либо **display: inline-flex** и любым значением **z-index**, кроме **auto**.

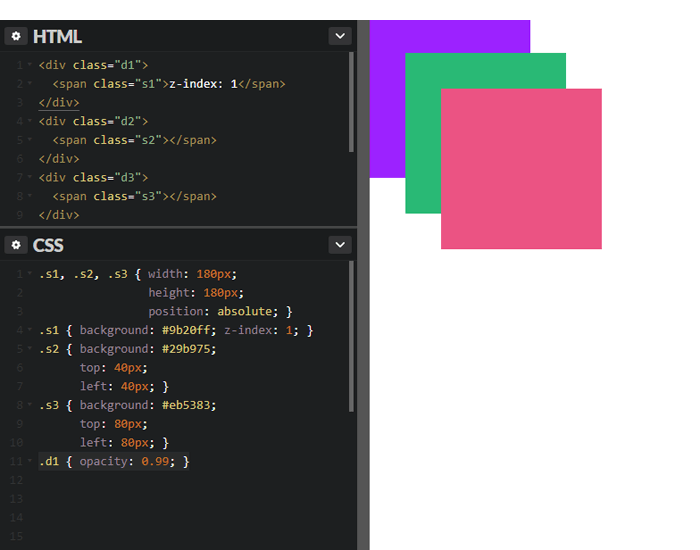
* Элемент со свойством **opacity** и значением менее **1**.
* Элемент с любым значением свойства **transform**, кроме **none**.
* Элемент с любым значением свойства **mix-blend-mode**, кроме **normal**.
* Элемент с любым значением свойства **filter**, кроме **none**.

Итак, если соблюдать один из вышеперечисленных пунктов (применить к элементу позиционирование и **z-index** либо свойство **opacity** со значением меньше единицы и т. п.), то формируется новый контекст наложения. Внутри контекста наложения дочерние элементы можно перемещать по оси Z в соответствии с обычными правилами.

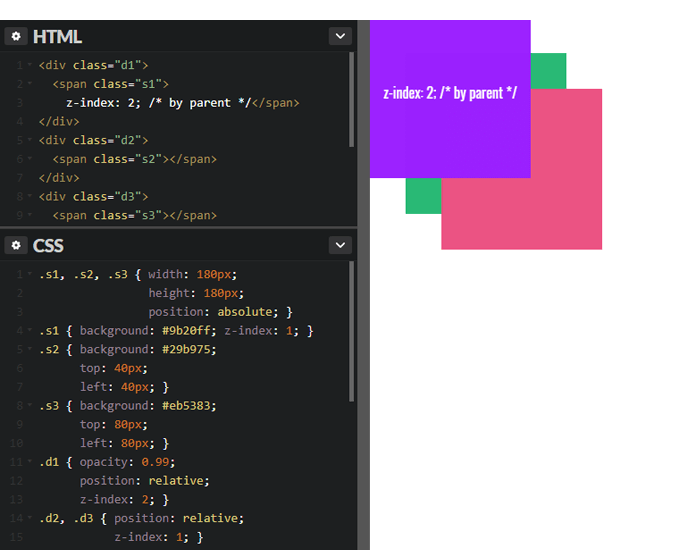
Контекст наложения может являться частью другого контекста наложения, тем самым создавая своеобразную иерархию контекстов. Если внутри родителя дочерний элемент создает свой собственный контекст наложения, то значения **z-index** будут учтены в пределах родителя. Элементы, которые не создают свой контекст наложения, используют родительский контекст наложения.

Один контекст наложения является полностью независимым от соседнего контекста. Это означает, что вы не можете, к примеру, наложить дочерний элемент **А** из одного контекста поверх дочернего элемента **Б** из другого контекста, если родитель элемента **А** находится ниже родителя элемента **Б** (подразумевается, что эти родители являются создателями разных контекстов).

Ниже показан пример того, как родительский элемент **.d1** создает новый контекст наложения при добавлении к нему свойства **opacity: 0.99**, после чего дочерний элемент **.s1** вновь становится нижним слоем, несмотря на свой **z-index**:



Это происходит потому, что теперь свойство **z-index** элемента **.s1** работает в пределах контекста наложения своего родителя **.d1**, тогда как другие два блока **<div>** пока имеют корневой контекст наложения. Каким же образом снова разместить фиолетовый блок выше других, учитывая свойство прозрачности? Для этого необходимо позиционировать все блоки **<div>**, после чего можно будет установить для них нужный порядок через **z-index**:



## В завершение

В начале урока мы сравнили принцип работы **z-index** со слоями в Adobe Photoshop. После того, как вы узнали о контекстах наложения, имеет смысл вспомнить и о группах слоев в Фотошопе. Представьте себе, что элемент со свойством z-index — это слой, а контекст наложения — это группа слоев. Вы можете как угодно изменять порядок наложения слоев в пределах группы. Также вы можете менять порядок наложения самих групп. Однако вы не можете наложить определенный слой из нижней группы на слои верхней группы — разве что переместить наверх всю нижнюю группу либо извлечь нужный слой из этой группы.

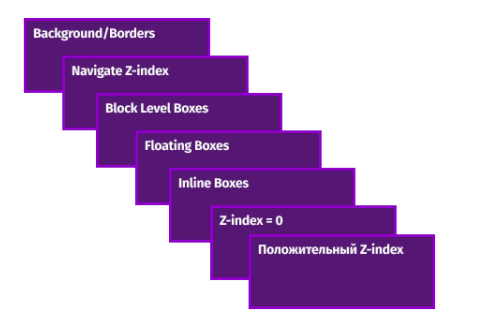
В примитивном варианте свойство z-index работает просто: чем больше значение, тем выше находится элемент (слой). Но стоит только столкнуться с разными контекстами наложения (группами), как всё становится намного сложнее, и начинает казаться, что z-index не работает. Рекомендуем дополнительно попрактиковаться в данной теме: создайте различные контексты наложения, используя список выше, и понаблюдайте за тем, как ведут себя элементы с z-index в этих контекстах. А в качестве удобной ассоциации вспоминайте слои и группы Adobe Photoshop.

[Порядок наложения элементов в CSS (Stacking Order)?](https://youtu.be/GZUy2i6QN7o?t=204)

Размещения элементов на странице определяется порядком наложения, который управляет приоритетом наложения одних элементов на другие. Создать свой порядок поможет свойство z-index.

Порядок наложения элементов

Если свойства z-index и position явно не заданы для элементов, контекст наложения определяется порядком их размещения в исходном коде.

****

Определяя значение свойства z-index мы и тем самым изменяем порядок контекста наложения, изменяя расположение с позиционированных элементов. Эти элементы будут представлены на странице в следующем порядке:

* корневой < html> элемент, который содержит все элементы веб-странице, а также фон и границы элементов;
* с позиционированные элементы с отрицательным значением z-index;
* не с позиционированные блочные элементы;
* не с позиционированные float элементы;
* не с позиционированные строковые элементы;
* позиционированные элементы со значениями z-index: 0 / auto;
* элементы с положительным значением z-index;

Cвойство z-index

Свойство z-index позволяет изменить порядок наложения позиционированных элементов в случае, когда они накладываются друг на друга.

Синтаксис z-index: 100; z-index: 120; z-index: -10; z-index: inherit; z-index: initial;

Значения

* auto(по умолчанию): присваивается значение 0. Если блок имеет фиксированное значение позиции или является корневым элементом, использование значения auto также создает новый контекст наложения.
* целое число: значение определяет расположение элемента в текущем контексте наложения, а также создает новый локальный контекст. Можно использовать любые целые числа, включая отрицательные. Отрицательные значения размещают элемент под элементами с нулевым и положительным индексами.
* initial: Устанавливает значение свойства в значение по умолчанию.
* inherit: Наследует значение свойства от родительского элемента.

Как упорядочить z-index внутри проекта? Перечислим несколько способов: .

* 1. Храните все значения z-index в одном месте, например в CSS / SCSS переменных. Тогда значениями z-index будет проще управлять.
* 2. Установите шкалу z-индекса, например, используя значения, кратные 100. Это поможет проще понять, какие элементы должны располагаться поверх других.
* 3. Избегайте использования произвольных значений z-индекса(таких, как «9999») поскольку они могут затруднить понимание наложение элементов.

[More](https://habr.com/ru/articles/166435/)

[Как с помощью CSS определить, поддерживается ли свойство в браузере?](https://youtu.be/70VnuTXi4Wk?t=32)

## С помощью одного CSS

Узнать, поддерживает ли данный браузер указанное CSS свойство, можно через CSS директиву под названием [*@supports*](https://developer.mozilla.org/ru/docs/Web/CSS/@supports).

Эта директива работает так же, как и @media.

/\* Узнаём, поддерживает ли браузер display: flex; \*/

@supports (display: flex) {

/\* Стили \*/

}

Так же можно установить свойства при условии, если браузер не поддерживает указанное CSS свойство

/\* Узнаём, поддерживает ли браузер display: flex; \*/

@supports not (display: flex) {

/\* Стили \*/

}

Ещё можно проверить несколько свойств сразу, используя оператор and

/\* Узнаём, поддерживает ли браузер display: flex; и flex-wrap: wrap; \*/

@supports (display: flex) and (flex-wrap: wrap) {

/\* Стили \*/

}

А если нам нужно хотя бы одно рабочее свойство, можно использовать оператор or

/\* Узнаём, поддерживает ли браузер display: flex; \*/

@supports (display: flex) or (display: -webkit-flex) {

/\* Стили \*/

}

Но нужно учесть, что данная директива новая и в IE вообще не работает.

Так же, для обеспечения, какой никакой, кроссбраузерности, можете использовать [вендорные префиксы](http://yapro.ru/web-master/css/vendornie-prefiksi.html).

[About JS](https://ru.stackoverflow.com/questions/710801/Как-узнать-о-поддержке-браузером-css-свойства" \l ":~:text=Узнать%2C поддерживает ли данный браузер,же%2C как и @media.)

[Как поддерживать страницы в браузерах с ограниченными функциями?](https://youtu.be/70VnuTXi4Wk?t=75)

[Как исправлять специфичные проблемы со стилями для разных браузеров?](https://youtu.be/70VnuTXi4Wk?t=122)

https://1ps.ru/blog/sites/2019/chto-takoe-krossbrauzernost-sajtov/

[Глобальные ключевые слова в CSS?](https://youtu.be/VYQl2GhbCUs?t=224)

https://doka.guide/css/global-keywords/

[Что такое CSS-атрибут (attr)?](https://youtu.be/VYQl2GhbCUs?t=301)

attr — это CSS-функция, которая умеет получать значение любого атрибута элемента, а потом использовать это значение прямо в стилях.

## Как пишется

Скопировать ссылку на секцию "Как пишется"

div::before {

content: attr(data-title);

content: attr(href);

С указанием типа:

div::before {

content: attr(src url);

content: attr(data-count number);

content: attr(data-width px);

С указанием фолбэка, то есть запасного значения:

div::before {

content: attr(data-count number, 0);

content: attr(src url, "");

content: attr(data-width px, inherit);

content: attr(data-something, "default");

## Подсказки

Скопировать ссылку на секцию "Подсказки"

💡 Функцию attr можно использовать в качестве значения любого CSS-свойства, однако полностью поддерживается только свойство [content](https://doka.guide/css/content/). Для остальных свойств поддержка экспериментальная и может различаться от браузера к браузеру. Актуальную информацию о поддержке можно посмотреть на [Can I use](https://caniuse.com/css3-attr).

💡 Написание с указанием типа или фолбэка пока имеет статус экспериментальной технологии и не поддерживается браузерами. Но в будущем это позволит гораздо сильнее расширить область применения функции attr. Например, мы сможем написать так:

.colored {

background-image: attr(data-src url);

}

Тут мы указали, что в качестве значения для свойства [background](https://doka.guide/css/background-image/) мы хотим использовать значение атрибута data. При этом уточнили, что значение атрибута data — это не просто строка, а корректный URL.

Примеры записи с указанием типа или фолбэка кажутся довольно перспективными, но, к сожалению, пока не поддерживаются ни одним из браузеров.

🛠 Самый распространённый случай использования функции attr - отображение значения атрибута href после текста ссылки при печати страницы.

<p>

Подробнее о скидках и акциях можно узнать

<a href="http://best-site.ru/sales">по ссылке</a>

</p>

[Что такое перечисление селекторов?](https://youtu.be/VYQl2GhbCUs?t=336)

При помощи перечисления нескольких селекторов через запятую можно избежать дублирования кода.

🤖 Если у нескольких элементов есть определённые повторяющиеся стили, то вполне допустимо вынести эти стили в отдельный блок и в качестве селектора указать несколько классов (или тегов, или идентификаторов), перечислив их через запятую.

🤖 Перечисляемые через запятую селекторы могут быть никак не связаны между собой родственными связями, могут находиться в разных частях разметки или вообще на разных страницах. Стили, описанные внутри фигурных скобок, будут в равной степени применены и к первому, и ко второму, и ко всем последующим селекторам.

Как пишется

Указывается один селектор, после него ставится запятая, через пробел указывается второй селектор и так далее. Для удобства чтения можно переносить каждый новый селектор на новую строку, но технически это не обязательно.

Селекторами может быть не только селектор по классу, но и селектор по тегу или по идентификатору, или даже их комбинация.

Подсказки

💡 Если стили у перечисляемых блоков не отличаются ничем, то это причина пересмотреть разметку и задать одинаковым по стилям элементам одинаковый класс.

💡 Если перечисляете через запятую составные селекторы, то нужно указывать их полностью.

🛠 Начинающие разработчики часто допускают ошибку при перечислении селекторов: по их логике первая часть составного селектора не требует повторения и её можно указать один раз в начале, а затем перечислить через запятую несколько «вторых» частей составного селектора.

Вложенным селектор считается только в первой строке, остальные классы отдельные, сами по себе.

[Для чего используется ключевое слово currentColor в CSS?](https://youtu.be/trriSYNrHw4?t=549)

Оно позволяет использовать значение color для свойств, которые не получают его по умолчанию. Если текущий цвет используется в качестве значения свойства color , он вместо этого принимает его значение из наследуемого значения свойства color .

<https://www.8host.com/blog/klyuchevye-slova-css-currentcolor/>

Ключевое слово current можно использовать в качестве значения для CSS-свойства, принимающего цвет. Например, [background](https://doka.guide/css/background-color/). Браузер подставит вместо current текущее значение свойства [color](https://doka.guide/css/color/).

Принцип работы очень похож на [кастомные свойства](https://doka.guide/css/custom-properties/) в CSS: берём переменную, присваиваем значение и используем в нужных нам местах. Отличие только в том, что мы не создаём отдельную переменную, а используем готовое ключевое слово.

Если кастомные свойства вам не знакомы, то есть другая хорошая аналогия — ключевое слово [inherit](https://doka.guide/css/global-keywords/" \l "inherit). Используем его в качестве значения нужного свойства и при изменении этого стиля у родителя стиль изменится сразу и в потомке. Недостаток такого подхода в том, что мы можем подставить это значение только в то свойство, которое нужно наследовать, а не в несколько.

Есть момент, о котором не нужно забывать. Если вы указали цвет текста и несколько свойств, где используется current, и потом переопределили [color](https://doka.guide/css/color/), то ко всем свойствам применится именно последний указанный цвет.

🛠 Есть очень хороший пример использования current. Допустим, есть кнопка с текстом и поясняющей иконкой внутри. Мы можем применить current для окрашивания иконки в такой же цвет, как и текст. В итоге, просто меняя свойство [color](https://doka.guide/css/color/), мы стилизуем оба элемента внутри кнопки.

Псевдокласс в CSS — это ключевое слово, добавленное к селектору, которое определяет его особое состояние. Например, [:hover](https://developer.mozilla.org/ru/docs/Web/CSS/:hover) может быть использован для изменения цвета кнопки при наведении курсора на неё.

div:hover {

background-color: #F89B4D;

}

Псевдоклассы дают возможность стилизовать элемент на основе не только отношений в DOM-дереве, но и основываясь на внешних факторах, таких как история посещений (например, [:visited](https://developer.mozilla.org/ru/docs/Web/CSS/:visited)), состояние содержимого (вроде [:checked](https://developer.mozilla.org/ru/docs/Web/CSS/:checked) у некоторых элементов формы) или позиции курсора мыши (например, [:hover](https://developer.mozilla.org/ru/docs/Web/CSS/:hover) определяет, находится ли курсор мыши над элементом).

Copy to Clipboard

В отличие от псевдоклассов, [псевдоэлементы](https://developer.mozilla.org/ru/docs/Web/CSS/Pseudo-elements) могут быть использованы для стилизации определённой части элемента.

## [Синтаксис](https://developer.mozilla.org/ru/docs/Web/CSS/Pseudo-classes" \l "синтаксис)

selector:pseudo-class {

property: value;

}

Как и с обычными классами, можно совмещать вместе в одном селекторе любое число псевдоклассов.

## [Список стандартных псевдоклассов](https://developer.mozilla.org/ru/docs/Web/CSS/Pseudo-classes" \l "список_стандартных_псевдоклассов)

* [:active](https://developer.mozilla.org/ru/docs/Web/CSS/:active)
* [:any (en-US)](https://developer.mozilla.org/en-US/docs/Web/CSS/:is)
* [:any-link](https://developer.mozilla.org/ru/docs/Web/CSS/:any-link)
* [:checked](https://developer.mozilla.org/ru/docs/Web/CSS/:checked)
* [:default](https://developer.mozilla.org/ru/docs/Web/CSS/:default)
* [:defined](https://developer.mozilla.org/ru/docs/Web/CSS/:defined)
* [:dir()](https://developer.mozilla.org/ru/docs/Web/CSS/:dir)
* [:disabled](https://developer.mozilla.org/ru/docs/Web/CSS/:disabled)
* [:empty](https://developer.mozilla.org/ru/docs/Web/CSS/:empty)
* [:enabled](https://developer.mozilla.org/ru/docs/Web/CSS/:enabled)
* [:first](https://developer.mozilla.org/ru/docs/Web/CSS/:first)
* [:first-child](https://developer.mozilla.org/ru/docs/Web/CSS/:first-child)
* [:first-of-type](https://developer.mozilla.org/ru/docs/Web/CSS/:first-of-type)
* [:fullscreen](https://developer.mozilla.org/ru/docs/Web/CSS/:fullscreen)
* [:focus](https://developer.mozilla.org/ru/docs/Web/CSS/:focus)
* [:hover](https://developer.mozilla.org/ru/docs/Web/CSS/:hover)
* [:indeterminate](https://developer.mozilla.org/ru/docs/Web/CSS/:indeterminate)
* [:in-range](https://developer.mozilla.org/ru/docs/Web/CSS/:in-range)
* [:invalid](https://developer.mozilla.org/ru/docs/Web/CSS/:invalid)
* [:lang()](https://developer.mozilla.org/ru/docs/Web/CSS/:lang)
* [:last-child](https://developer.mozilla.org/ru/docs/Web/CSS/:last-child)
* [:last-of-type](https://developer.mozilla.org/ru/docs/Web/CSS/:last-of-type)
* [:left](https://developer.mozilla.org/ru/docs/Web/CSS/:left)
* [:link](https://developer.mozilla.org/ru/docs/Web/CSS/:link)
* [:not()](https://developer.mozilla.org/ru/docs/Web/CSS/:not)
* [:nth-child()](https://developer.mozilla.org/ru/docs/Web/CSS/:nth-child)
* [:nth-last-child()](https://developer.mozilla.org/ru/docs/Web/CSS/:nth-last-child)
* [:nth-last-of-type()](https://developer.mozilla.org/ru/docs/Web/CSS/:nth-last-of-type)
* [:nth-of-type()](https://developer.mozilla.org/ru/docs/Web/CSS/:nth-of-type)
* [:only-child](https://developer.mozilla.org/ru/docs/Web/CSS/:only-child)
* [:only-of-type](https://developer.mozilla.org/ru/docs/Web/CSS/:only-of-type)
* [:optional](https://developer.mozilla.org/ru/docs/Web/CSS/:optional)
* [:out-of-range](https://developer.mozilla.org/ru/docs/Web/CSS/:out-of-range)
* [:read-only](https://developer.mozilla.org/ru/docs/Web/CSS/:read-only)
* [:read-write](https://developer.mozilla.org/ru/docs/Web/CSS/:read-write)
* [:required](https://developer.mozilla.org/ru/docs/Web/CSS/:required)
* [:right](https://developer.mozilla.org/ru/docs/Web/CSS/:right)
* [:root](https://developer.mozilla.org/ru/docs/Web/CSS/:root)
* [:scope (en-US)](https://developer.mozilla.org/en-US/docs/Web/CSS/:scope)
* [:target](https://developer.mozilla.org/ru/docs/Web/CSS/:target)
* [:valid](https://developer.mozilla.org/ru/docs/Web/CSS/:valid)
* [:visited](https://developer.mozilla.org/ru/docs/Web/CSS/:visited)

https://www.w3.org/wiki/CSS3/Selectors

[Какие фильтры есть в CSS?](https://youtu.be/trriSYNrHw4?t=662)

https://snipp.ru/html-css/css-filter

[Для чего используется псевдокласс :invalid?](https://youtu.be/xIGp2FCxqj0?t=370)

Описание CSS псевдокласс :invalid находит любые <input> или <form> элементы, контент которых не проходит валидацию (en-US), в соответствии с типом поля. Он позволяет вам легко менять внешний вид полей, что позволяет пользователю видеть и исправлять ошибки.

[Расскажите про свойство display в CSS?](https://youtu.be/xIGp2FCxqj0?t=436)

Свойство display ([CSS](https://developer.mozilla.org/ru/docs/Web/CSS)) определяет тип отображения (display type) элемента, имеющий два основных свойства, определяющих генерацию боксов — внешний тип отображения определяет расположение бокса в [схеме потока (flow layout)](https://developer.mozilla.org/ru/docs/Web/CSS/CSS_Flow_Layout) и внутренний тип отображения определяет расположение дочерних элементов бокса (бокс - это прямоугольная область, являющаяся изображением элемента).

Некоторые значения свойства display полностью определены в их индивидуальных спецификациях; смотрите таблицу в конце этого документа со ссылками на все релевантные спецификации. Полный список значений приведён ниже.

|  |  |
| --- | --- |
| [Начальное значение](https://developer.mozilla.org/ru/docs/Web/CSS/initial_value) | inline |
| Применяется к | все элементы |
| [Наследуется](https://developer.mozilla.org/ru/docs/Web/CSS/Inheritance) | нет |
| [Обработка значения](https://developer.mozilla.org/ru/docs/Web/CSS/computed_value) | как указанное значение, кроме как для позиционированных и плавающих элементов и корневого элемента. В обоих случаях вычисляемое значение может быть ключевым словом, отличным от указанного. |
| Animation type | Not animatable |

## [Синтаксис](https://developer.mozilla.org/ru/docs/Web/CSS/display" \l "синтаксис)

Свойство display задаётся с помощью ключевых слов. Ключевые слова группируются по шести категориям:

[<display-outside>](https://developer.mozilla.org/ru/docs/Web/CSS/display-outside)

Эти ключевые слова определяют внешний тип отображения (outer display type) элемента, который играет существенную роль в схеме потока.

[<display-inside> (en-US)](https://developer.mozilla.org/en-US/docs/Web/CSS/display-inside)

Эти ключевые слова определяют внутренний тип отображения ( inner display type) элемента, который определяет тип контекста форматирования, в котором располагается его содержимое (при условии, что это незаменяемый элемент).

[<display-listitem>](https://developer.mozilla.org/ru/docs/Web/CSS/display-listitem)

Создаёт блоковый бокс для контента и отдельный строковый (inline) бокс для элемента списка.

[<display-internal> (en-US)](https://developer.mozilla.org/en-US/docs/Web/CSS/display-internal)

Некоторые модели разметки, такие как table и ruby, имеют сложную внутреннюю структуру с несколькими различными ролями, которые могут выполнять их дочерние элементы и потомки. Этот раздел определяет те "внутренние" значения отображения, которые имеют смысл только в рамках этих конкретных методов разметок.

[<display-box> (en-US)](https://developer.mozilla.org/en-US/docs/Web/CSS/display-box)

Эти значения определяют, генерирует ли элемент отображение боксов вообще.

[<display-legacy> (en-US)](https://developer.mozilla.org/en-US/docs/Web/CSS/display-legacy)

В CSS 2 используется синтаксис с одним ключевым словом для свойства display, для которого требуются отдельные ключевые слова для блокового и строкового уровней одного способа разметки.

[<display-xul>](https://developer.mozilla.org/ru/docs/Mozilla/Gecko/Chrome/CSS/display-xul)

Non-standard

Устарело

Значения, используемые только в Firefox, в основном, для стилизации. [XUL documents](https://developer.mozilla.org/ru/docs/Mozilla/Tech/XUL).

### ["Наследственные" значения отображения](https://developer.mozilla.org/ru/docs/Web/CSS/display" \l "наследственные_значения_отображения)

Спецификация уровня 3 подразумевает два значения для свойства display — позволяет специфицировать внешний и внутренний тип отображения явно — но это поддерживается браузерами пока недостаточно хорошо.

Методы display-legacy позволяют получать такие же результаты с одиночными значениями ключевых слов и должны поддерживаться разработчиками до тех пор, пока два значения ключевых слов не будут лучше поддерживаться. Например, используя два значения, вы можете задать строковый (inline) flex контейнер следующим образом:

.container {

display: inline flex;

}

В настоящее время это можно задать с помощью одного значения.

Copy to Clipboard

.container {

display: inline-flex;

}

## [Доступность](https://developer.mozilla.org/ru/docs/Web/CSS/display" \l "доступность)

### [display: none;](https://developer.mozilla.org/ru/docs/Web/CSS/display" \l "display_none)

Если свойство display принимает значение none на элементе, то элемент удаляется из [дерева доступности](https://developer.mozilla.org/ru/docs/Learn/Accessibility/What_is_accessibility). Это приводит к тому, что элемент и все его дочерние элементы больше не будут восприниматься технологиями чтения экрана.

Если вы хотите визуально скрыть элемент, более доступной альтернативой является использование [комбинации свойств](_blank) для визуального удаления изображения с экрана, но это сохраняет его для синтаксического анализа с помощью вспомогательных технологий, таких как считыватели экрана.

### [display: contents;](https://developer.mozilla.org/ru/docs/Web/CSS/display" \l "display_contents)

Браузеры удаляют любой элемент со свойством display, имеющим значение contents из [дерева доступности](https://developer.mozilla.org/ru/docs/Learn/Accessibility/What_is_accessibility). Это приводит к тому, что элемент (но не его дочерние элементы) больше не будут восприниматься технологиями чтения экрана. Дочерние элементы становятся дочерними элементами элемента следующего уровня в DOM.

* [Display: Contents Is Not a CSS Reset | Adrian Roselli](https://adrianroselli.com/2018/05/display-contents-is-not-a-css-reset.html)
* [More accessible markup with display: contents — hiddedevries.nl](https://hiddedevries.nl/en/blog/2018-04-21-more-accessible-markup-with-display-contents)

### [Таблицы](https://developer.mozilla.org/ru/docs/Web/CSS/display" \l "таблицы)

Если у элемента [<table> (en-US)](https://developer.mozilla.org/en-US/docs/Web/HTML/Element/table) изменить значение свойства display на block, grid или flex, это изменит его представление в [дереве доступности](https://developer.mozilla.org/ru/docs/Learn/Accessibility/What_is_accessibility). Это приводит к тому, что таблица не будет объявлена должным образом с помощью технологии чтения экрана.

https://html5css.ru/cssref/pr\_class\_display.php

[В каком случае лучше использовать translate() вместо абсолютного позиционирования?](https://youtu.be/xIGp2FCxqj0?t=543)

Если к элементу применяется анимация или плавные переходы, у translate есть преимущества в производительности. Если в переходах или keyframe анимации есть этапы с передвижением элементов, лучше используйте translate. Касается как абсолютного, так и относительного позиционирования.

[Answer](https://webformyself.com/css-ot-a-do-ya-raznica-mezhdu-translate-i-position-relative/" \l ":~:text=Если к элементу применяется анимация,абсолютного%2C так и относительного позиционирования.)

[Что такое плавающие элементы (floats)? Как они работают?](https://youtu.be/xIGp2FCxqj0?t=610)

CSS-свойство float указывает, что текущий элемент должен быть изъят из обычного flow (потока) и прижат к левой или правой стороне родительского элемента. Текст и inline элементы будут обтекать такой элемент. Плавающий элемент - это любой элемент со свойством float отличным от none .

[More info incl. Clearing](https://developer.mozilla.org/ru/docs/Web/CSS/float)

[Расскажите о свойстве text-rendering?](https://youtu.be/nbWY5W-9OEo?t=35)

Свойство text позволяет указывать браузеру, какие оптимизации нужно производить с текстом во время рендера.

https://doka.guide/css/text-rendering/

[Расскажите о свойстве text-decoration-skip-ink?](https://youtu.be/nbWY5W-9OEo?t=156)

Свойство text-decoration-skip-ink управляет внешним видом верхнего и нижнего подчёркиваний. Если текст нужно перечеркнуть, то свойство text-decoration-skip-ink будет проигнорировано.

[Answer](https://doka.guide/css/text-decoration-skip-ink/" \l ":~:text=Свойство text-decoration-skip-,-skip-ink будет проигнорировано.)

[Расскажите о свойстве pointer-events?](https://youtu.be/J6CgOSKFOlw?t=336)

Свойство pointer-events позволяет управлять тем, как элементы будут реагировать на события мыши или прикосновения к сенсорному экрану. Часто применяется для того, чтобы в сложной компоновке можно было взаимодействовать с нижележащими элементами, игнорируя вышележащие.

[More](https://developer.mozilla.org/ru/docs/Web/CSS/pointer-events)

[Расскажите о свойстве outline?](https://youtu.be/3NGkctg4lsE?t=498)

Свойство outline задаёт дополнительную рамку вокруг элемента, за пределами его CSS-блока. Поддерживается во всех браузерах, IE8+. В отличие от border , рамка outline не участвует в блочной модели CSS. Она не занимает места и не меняет размер элемента.

Что-то вроде border на стероидах. Не влияет на размеры элемента, можно отодвинуть от границ на любое расстояние.

Присутствует в стандартных стилях браузера для акцента на сфокусированных интерактивных элементах. Типа размеры не меняются, вёрстка страницы не ломается, но пользователь наглядно видит, какой элемент в фокусе. Ситуация win-win!

[More](https://doka.guide/css/outline/)

[Расскажите о свойстве scrollbar-gutter?](https://youtu.be/DQ0BLu6rZYc?t=29)

scrollbar-gutter решает проблему с «прыгающей» шириной контента при появлении или скрытии скроллбара. Обычно эта проблема появляется при изменении высоты контента или появлении модального окна.

[More](https://doka.guide/css/scrollbar-gutter/" \l ":~:text=scrollbar-gutter решает проблему с,контента или появлении модального окна.)

[Почему не стоит использовать краткую запись свойств CSS?](https://youtu.be/3NGkctg4lsE?t=404)

[Answer](http://prgssr.ru/development/kratkaya-zapis-css-kak-antipattern.html" \l ":~:text=Из этого факта следует один,CSS намного сложнее%2C чем добавлять.)

[Назовите псевдоэлементы для подсветки текста?](https://youtu.be/3NGkctg4lsE?t=584)

Selection?

[Способы задания цвета в CSS?](https://youtu.be/hL5yFo9Pms4?t=384)

[Цвета и единицы измерения](https://loftschool.com/blog/posts/7-cvetovye-modeli-i-edinicy-izmereniya/)

**CSS Grid and Flexbox**

**Flexbox**

[A good article](https://tproger.ru/translations/how-css-flexbox-works/)

[MDN tutorial](https://developer.mozilla.org/en-US/docs/Learn/CSS/CSS_layout/Flexbox)

[w3schools](https://www.w3schools.com/css/css3_flexbox.asp)

[Shpargalka](https://tpverstak.ru/flex-cheatsheet/)

<https://metanit.com/web/html5/12.1.php>

**Grid**

<https://metanit.com/web/html5/13.1.php>

[A good article](https://habr.com/ru/companies/macloud/articles/564182/)

[MDN tutorial](https://developer.mozilla.org/en-US/docs/Web/CSS/CSS_Grid_Layout/Basic_Concepts_of_Grid_Layout)

[W3shools](https://www.w3schools.com/css/css_grid.asp)

[Shpargalka](https://tpverstak.ru/grid/)