Продолжение **html.**

Создание таблиц. Начало **CSS.**

**CSS –** стили, нужны для создания внешнего вида и расположения элементов, для дизайна.

(Каскадные таблицы стилей. Cascade Style Sheet) Просто дословный перевод.

Разумеется, у него свой синтаксис и спецификации.

**CSS** обычно подключается из другого документа, но есть способ прямо внутри **HTML.** Так называемые **встроенные стили.** Начнем с них.

Их помещают внутри **<head>**

**Пример:**



То к чему будет применяться стиль настраивается с помощью селектора. Они бывают разные. Но как я понимаю, на картинке выше **p** это селектор и подключенный таким образом стиль применится ко всем тегам **<p>** в **html.** Попробую.

Понятно, да добавив в **head** этот стиль, все что у меня было под тегом **<p>** стало красным.

Я даже попробовал написать еще один селектор, назвал его **marok** и ниже написал тег **<marok>** в который написал некий текст и на этот тест применился этот стиль.

Так официально, что такое **селектор?**

**Селектор элемента –** это селектор с названием тега. Его стиль применится ко всем элементам, расположенным под этим тегом. Еще раз для **P** ко всем **параграфам.**

**Color: цвет**

**Font-size: размер шрифта.**

**: 300px пиксели.** В отличие от html надо писать не просто цифры, а добавить **px.**

Еще **font-size** пишут в каких-то **em,** пока не знаю, что это.

Так вот, можно нажать f12, затем нажать на элемент и в инструментах разработчика будут те стили, которые относятся к этому элементы.

**Font-weight** толщина шрифта. Без **px. Просто цифры.**

Другой способ подключения **css – Инлайновый способ.** Таким способом мы пишем стиль прямо внутри элемента.

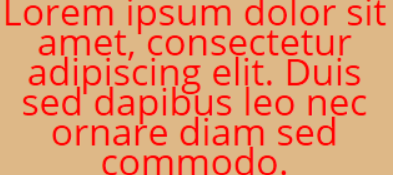
Стиль написания чуть отличается. Стиль закрывать не надо. И нет фигурных скобок. Все пишется внутри кавычек. А **style** выступает в качестве атрибута.

<h1 style="font-size: 100px; color: yellow;">Gusto</h1>

В этой структуре важны **кавычки** и **пробел** после **двоеточия.**

Рассмотрим несколько базовых свойств:

**Background-color:** цвет фона элемента.



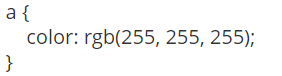
**font-weight:** сделать текст жирным или полужирным. Можно использовать такие ключевые слова:

**normal, bold.**

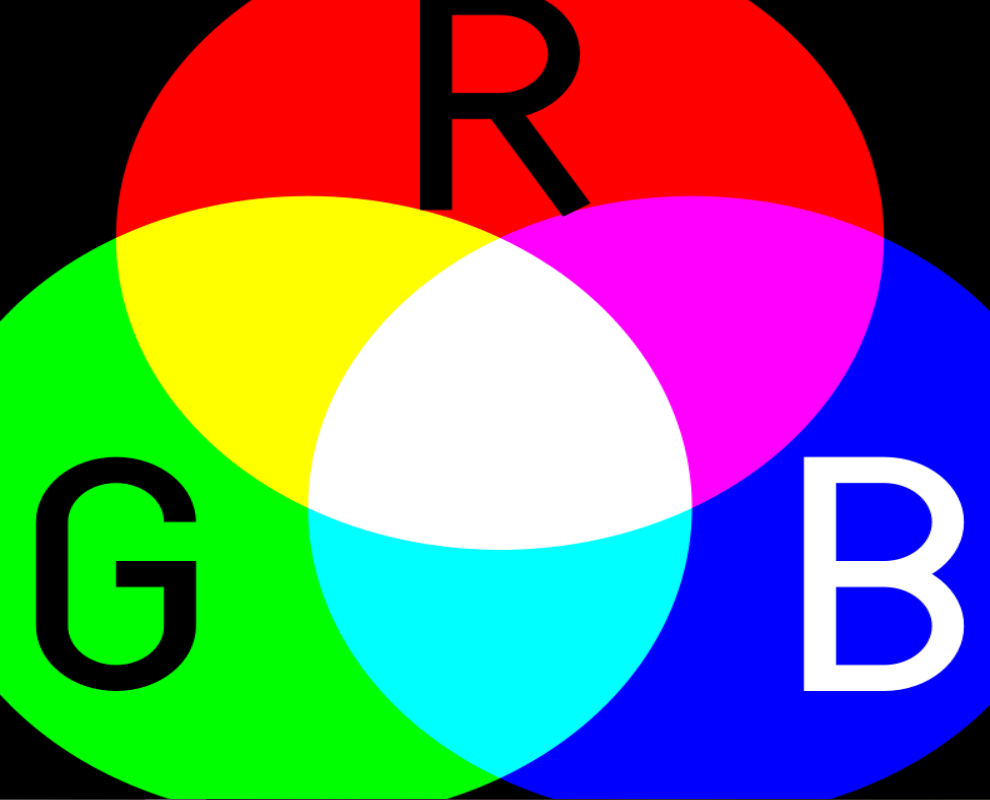
Цвета бывают разные, например: **color: tomato;**

**Работа с цветом.**

Представить цвет с помощью цифр. Например, в формате **RGB.**



Так их можно совмещать.



Чуть сложнее, но можно цвету также создавать полупрозрачность, это еще один способ задать цвет через **a – альфа канал**, этот параметр может колебаться в промежутке от 0 до 1. Где 0 **полностью прозрачный** и 1 **непрозрачный.**

**Пример черного полупрозрачного цвета:**

**Color: rgba(0 ,0 ,0, 0.5);**

В стандартной **rgb** модели насчитывают 2^24 цветов, то есть 16 777 216 вариантов. 24 бита. Но параметром альфа 32 бита. 4 миллиарда.

В **RGB** есть и 16 представление цвета, его сложнее запомнить, но легче скопировать. **Пример:**

**color: #000000; черный**

**color: #ffffff; белый**

Есть и короткая форма записи:

**color: #000; черный**

**color: #fff; белый**

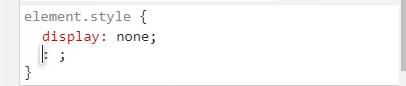
Можно самому подбирать или использовать готовые палитры и с их помощью находить нужный цвет. Эти палитры можно даже встроить в редакторы, в которых пишем код. **Color picker.**

Цвет также удобно подобрать прямо в инструментах разработчика. Это делает если нажать на этот квадратик у цвета:



В коде часто встречаются такие элементы как **<div>** Они не несут какого-то смыслового значения и используются для стилей. Им оборачивают какие-нибудь элементы и этому элементу задаем какие-то стили. Или использовать для выравнивания, для раскладки на странице.

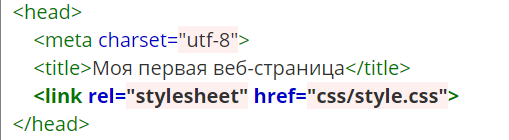
У **css** есть такое важно свойство, как **display,** если его выставить на **none,** и все, что было обернуто этим div исчезнет.



**Подключение CSS к проекту.**

Уже известно два способа подключения стилей, через **<style>** внутри **head** или инлайновый прямо внутри элемента. Но чаще всего используется третий, через подключение **стороннего файла.**

Он подключается внутри **head,** вот пример:



**Rel=”stylesheet”** и ссылка на расположение файла.

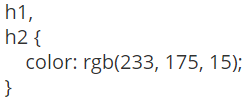
Если оформление на нескольких страницах одинаковое, то можно для всех использовать 1 css иначе несколько файлов. Стиль после подключения остается в браузере, тем самым ускоряя его работоспособность при загрузке.

Можно подключить несколько стилей, но если есть свойства с одинаковыми назначением, тот стиль, что был подключен последним перезапишет предыдущие свойства. **Пример:**



/\* Комментарий в CSS \*/

Свойства одного правила можно применять к **нескольким селекторам**. **Пример:**



**Селектор класса.**

Помню я давал секторам свои названия и все работало. Но до этого пока не дошли. Допустим нужно двум заголовкам <h2> привязать разные свойства, но это сложные свойства и их не удобно прописывать для каждого внутри. Тогда проще будет воспользоваться **селектором класса. Через атрибут класс.**



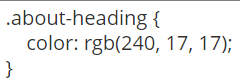
Название может быть любым, но есть правила:

**Называть понятно.**

**Не использовать английский**

**Вместо пробелов использовать – тире**

А вот так, он оформляется в самом **css.** Нужно поставить в начале **точку**, а затем название. **Пример:**



Класс можно применять к любому элементу в **body.**

**Font-weight:** можно использовать не только слова **normal, bold,** но и числа: **400, 700**(аналогичные параметры).

**Существуют уникальные селекторы:**

**Селектор идентификатора (id) –** используют в интерактивных формах и для **JavaScript.**

**Селектор всех элементов на странице (\*)** – его стоит использовать в редких случаях, например, изменение значения свойства **box-sizing** для удобства задания ширины всех элементов.

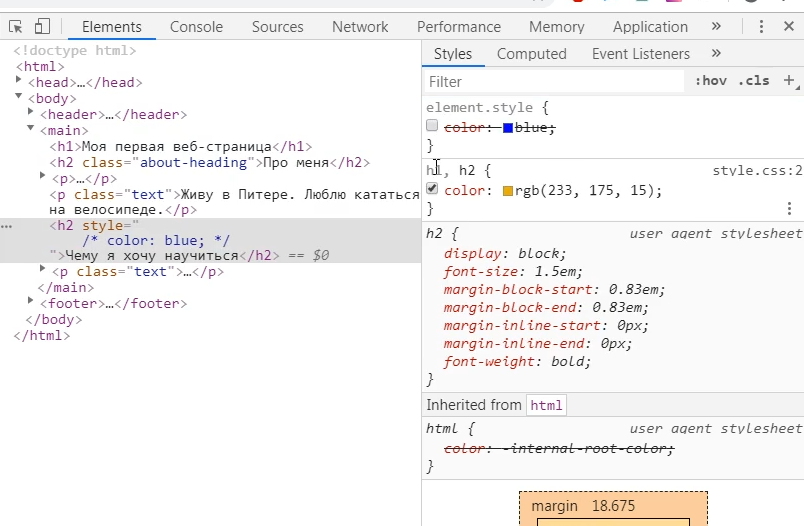
**Отношения:** читая справа на лево.

**Селектор селектор** – например **header a –** выберет все ссылке внутри шапки.

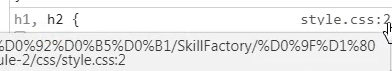
**Селектор > селектор –** непостредственные потомки на один уровень вложенности, **header > a** сработает также, а **main > strong** не сработает если **strong** всегда в **<p>.** Только 1 уровень вложенности.

**Селектор + селектор –** первый сосед. Выберет следующий элемент, на том же уровне. Например, если выбрать **h1 + p,** то этот стиль применится к первому параграфу после всех заголовков **h1.** Но к остальным параграфам не применится.

В инструментах разработчика в разделе **filter** можно прописать стиль, и он ляжет выше тех, что прописаны в **css(наверно? Да)**. И он инлайновым способом добавится в код.



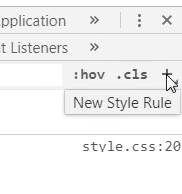
В инструменте разработчика даже будет показано в какой строке расположено начало этого правила.



Кроме того, на сайте можно увидеть **css,** которого нет в коде, так называемый встроенный **css.**

**user agent stylesheet**

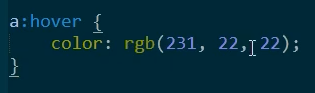
В инструментах разработчика можно на кнопку + добавить свои новые правила:

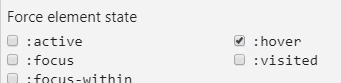


Такое правило будет описано не в коде, а так и написано, в инструменте разработчика.

**inspector-stylesheet**

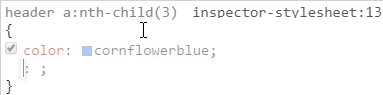
Кнопка **hov**  отвечает за состояние. Например, когда наводим мышкой на ссылку, она меняет цвет. **Пример:**





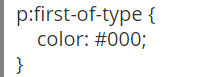
А тут можно имитировать различные состояния. Например, тот самый **hover**, это наведение.

Стили можно обращать к **потомкам**, например, такой стиль для третьей ссылки в header. (Это кстати псевдокласс)



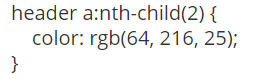
**Псевдоклассы.**

Содержимое страницы может меняться. (Кто-то, что-то добавит на сайт). Иногда проще указать не сам элемент, а его порядок. Например, самый первый или все четные и т.д. Для этого используют **псевдоклассы**. Их пишут через **двоеточи**е после **селектора. Пример:**

Это похоже применится только к первому **<P>**

Тот, что выше это :**first-of-type –** выберет первый элемент такого типа, внутри родительского элемента. Если несколько родителей с параграфами, то выберуются все первые параграфы.

**:nth-child() –** выберет элемент, если он определенный потомок. В скобках можно написать номер потомка или специальную формулу. **Пример:**



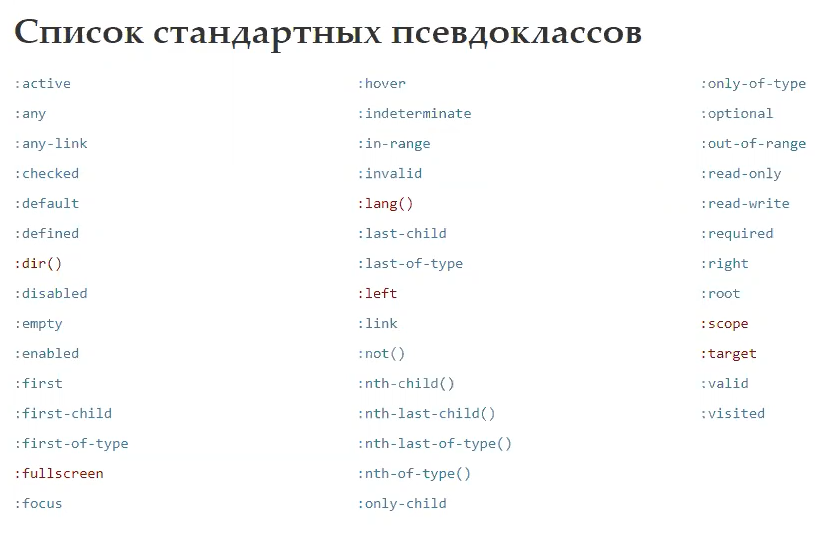
**:nth-of-type – сложно, в голове не укладывается, поэтому просто скопирую текст.** Похожим образом работает псевдокласс :nth-of-type. Только он выбирает потомка своего родителя: разница не всегда очевидна. Она проявляется, если у родителя есть другие потомки. На нашей странице есть два элемента h2, допустим нам нужен второй. Но, если мы попробуем использовать :nth-child, выберется первый, ведь именно он второй потомок родителя.

У элементов бывают разные состояния, которые можно **стилизовать.** Для этого нужно добавить к селектору псевдокласс состояния. Например, уже известный нам **:hover** **(при наведении).**

У ссылок есть особое состояние для **непосещенных (:link)** и **посещенных (:visited).**

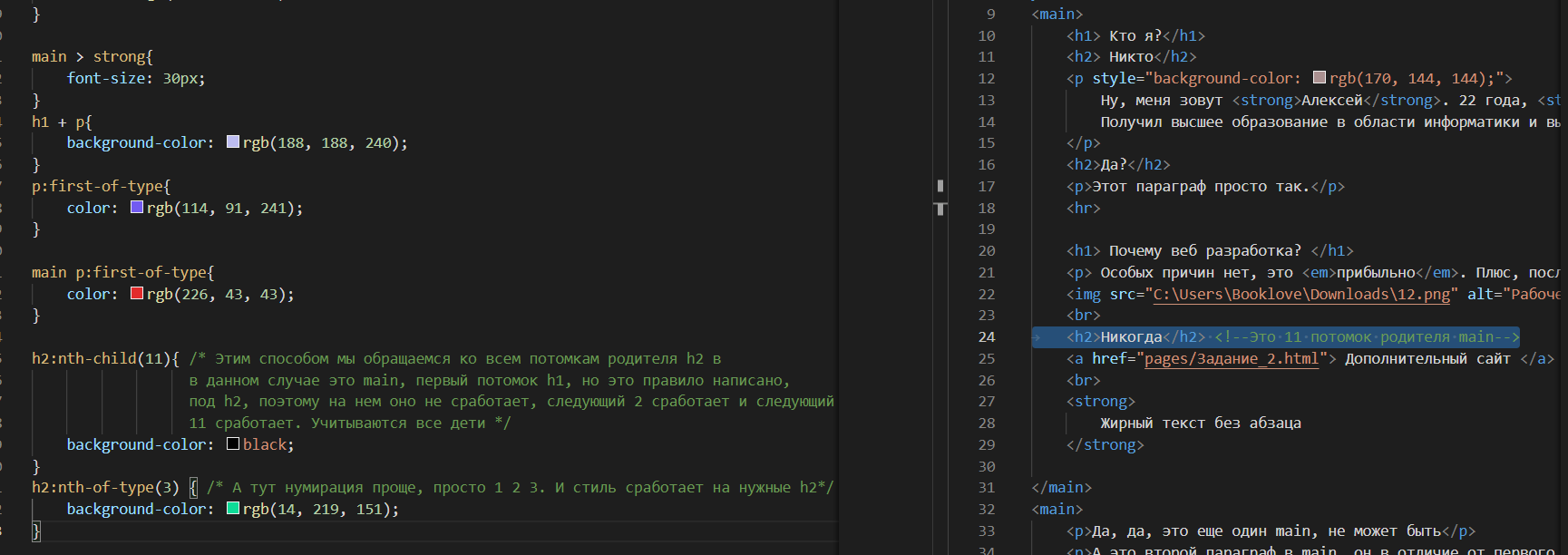
По умолчанию синий и фиолетовый цвет, но можно изменить.

**Видео:**



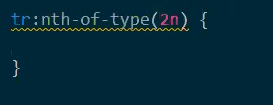
**Active** когда нажал на что-то.

Разницу понял, вот она:



В каком-то динамическом проекте можно использовать определенный стиль на последнего потомка, чтобы он всегда отличался, сколько бы их не добавлялось.

**Пример формулы:**

**Только четные потомки типа tr.**

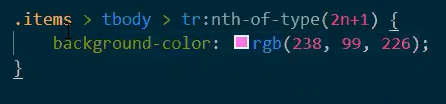
Для нечетных формула будет (**2n+1)**

Если схватило лишнего, то можно уточнить, где именно нужно.



Или добавить **>** чтобы указать, что только непосредственные **потомки.**

Можно добавить еще вложенность, чтобы это относилось только к конкретно таблице, но такого лучше избегать.

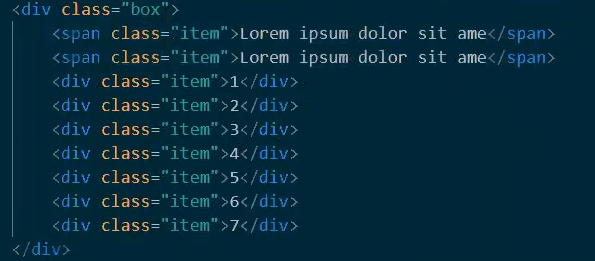


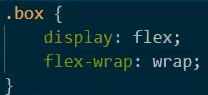
**Еще формулы:**

**(n+3)** Все, кроме первых трех.

**(-n+3)** Только первые три.

Вот маленькая галерея. И если на клас item применить **nth-of-type(2),** то этот стиль сработает на второй **span** и второй **div.** Поэтому, если в элементе расположены разные элементы, то лучше использовать **nth-child(2)**





**Верстка табличной информации.**

Все прекрасно понимают, что такое таблица.



Для таблицы важно, чтобы она использовалась для связанных с друг-другом данных.

**Чего делать таблицей не стоит -** раскладку (верстку основных блоков) страницы. Многие сайты для верстки используют таблицы. Сейчас инструментов **CSS** достаточно, так что для этих целей таблицы не использовать.

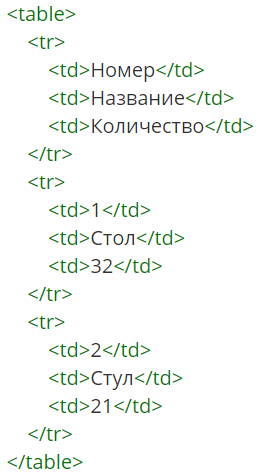
**Основные элементы:**

**<table> </table> -** таблица.

**<tr> </tr> -** ряд

**<td> </td> -** ячейка.

Пример.



Если открыть код в инструменте разработчика, то можно заметить различие от реального кода, добавится некий элемент: **<tbody>**

**DOM** отличается от реального кода



По умолчанию всегда будет создаваться элемент **<tbody>** т.е. тело таблицы. У нее также может быть шапка. Элементы внутри шапки принято помечать элементом **<th>** по умолчанию, эти элементы становятся **жирными.**

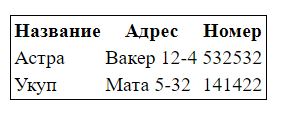
У таблицы нет границ по умолчанию, но к ней по-прежнему можно применять стили и делать с ней много чего.



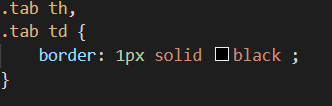
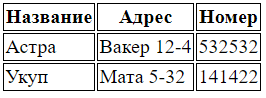
В традиционном виде таблицы должны содержать границы, но в современном вебе обходятся без них. По крайней мере для визуально простых таблиц.

Включим в стилях границы. Через свойство **border. Пример:**

**border: 1px solid black**

Это свойство применилось для всей таблицы.

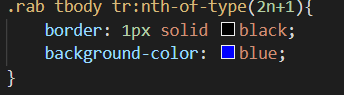
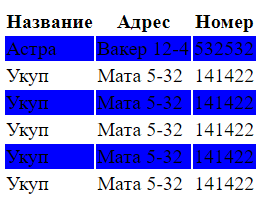
Аналогично нужно сделать для ее содержимого.

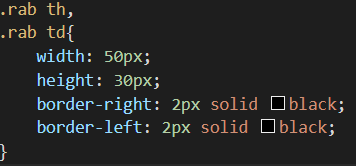
Внешнюю рамку заранее убрал. Сам **.tab** подключен к таблице, **solid:** цвет.



Вот так можно сделать, чтобы только нечетные строки таблицы были бы синими, причем в коде есть уточнение, чтобы **thead** не включался, а был бы только **tbody**. Вместо формулы (2n+1) можно было использовать **odd** или **even** для четных.

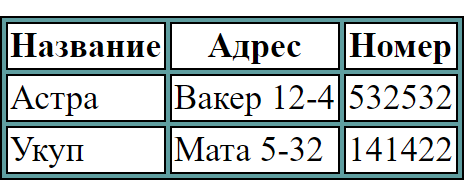
 

По умолчанию ячейки занимают столько место, сколько у них содержимое. Чтобы таблица была не слишком маленькой, можно задать ей размер в стилях. И также отдельно задать ей границы

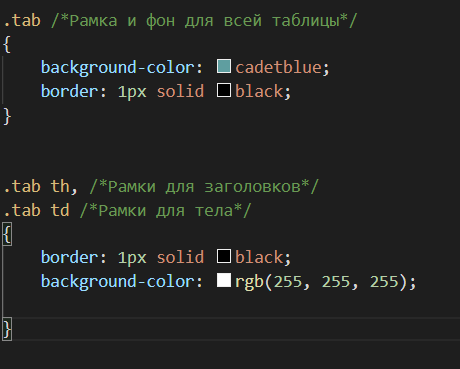
 

Но по середине получились двойные, мне это не нравится.

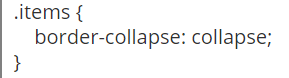
Я попробовал сделать вот такую таблицу. С помощью такого кода.



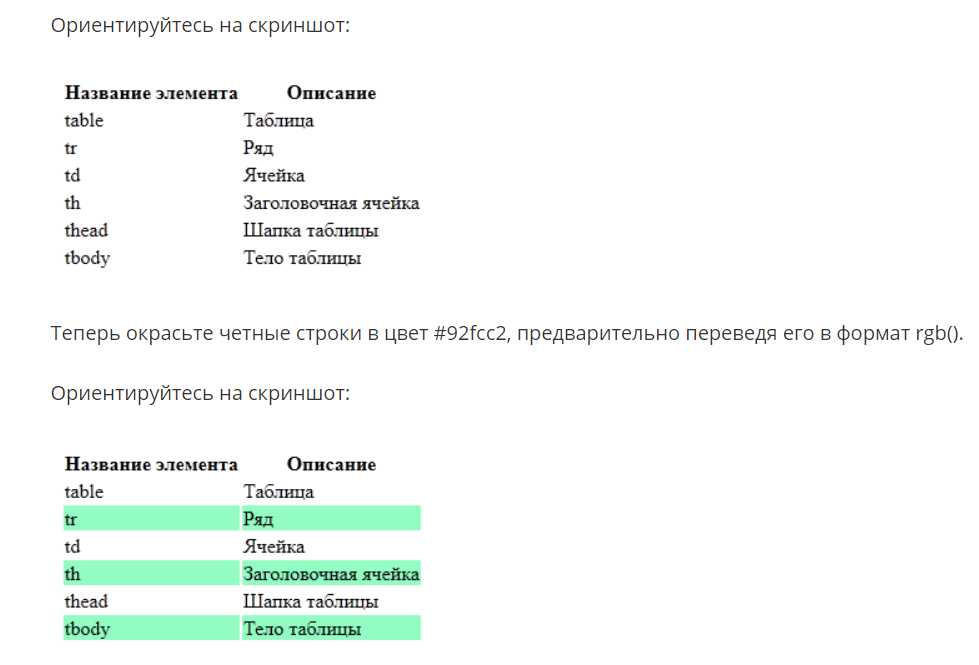
Такую таблицу можно сделать таким стилем:



Ой, случайно пропустил важную часть. Есть свойство, которые отвечает за отступ между ячейками – **border-collapse.** Вот как его отключить.



Не уверен, что это обязательно. Но сделаю это задание для практики.



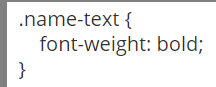
Все, сделал. Забыл подключить стили, а так нормально.

**Стилизация Span.**

Насколько я помню, он для стилизации отдельных частей в тексте. Одного или пары слов.

Понятно дело нельзя просто накидать кучу **span’ов,** а потом задать им стиль. Очевидно, что к нему добавляются **псевдоклассы.**

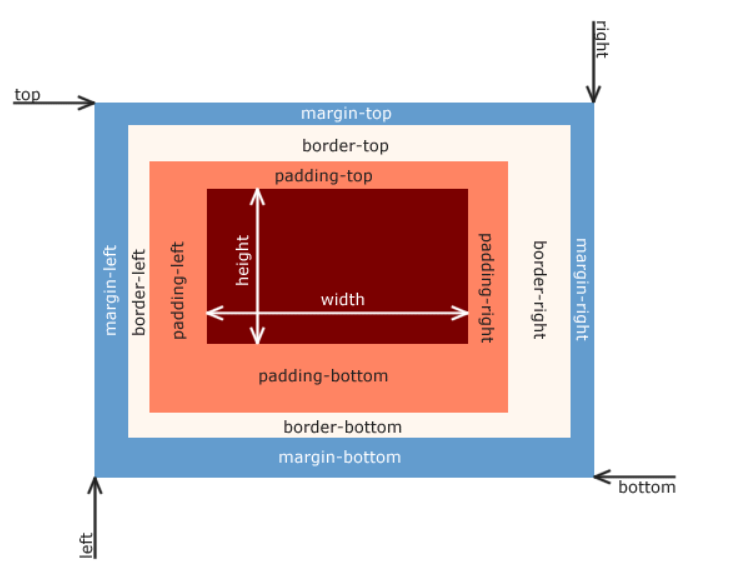




**font-style: italic;** Надо уже запомнить, что **border** рамка, а **font** – шрифт. **Italic – курсив.**

**Свойство дисплей.**

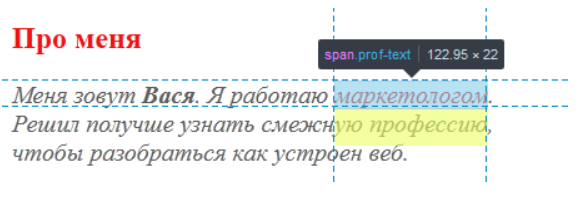
Свойство **display** нужно, чтобы раскладывать элементы по странице. Это расположение в соответсвии с тем, что называется **блочной моделью (Box model).**



**Box model** делает из элементов прямоугольник с параметрами ширины, высоты и отступами.

У **display** есть три базовых значения: **inline, block, inline-block.**

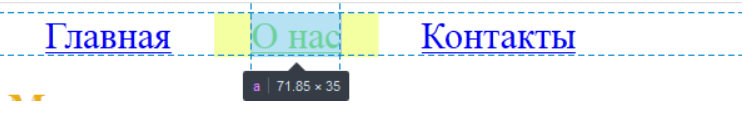
**Inline** – это строчный бокс, имеется у текстового содержимого параграфа, или заголовков. Размеры строчных боксов определяются исключительно содержимым. Нельзя изменить ширину, высоту или задать отступы. Например, на скриншоте у **span** задан нижней отступ “**ую профессию”,** но это не влияет на другое строчное содержимое:



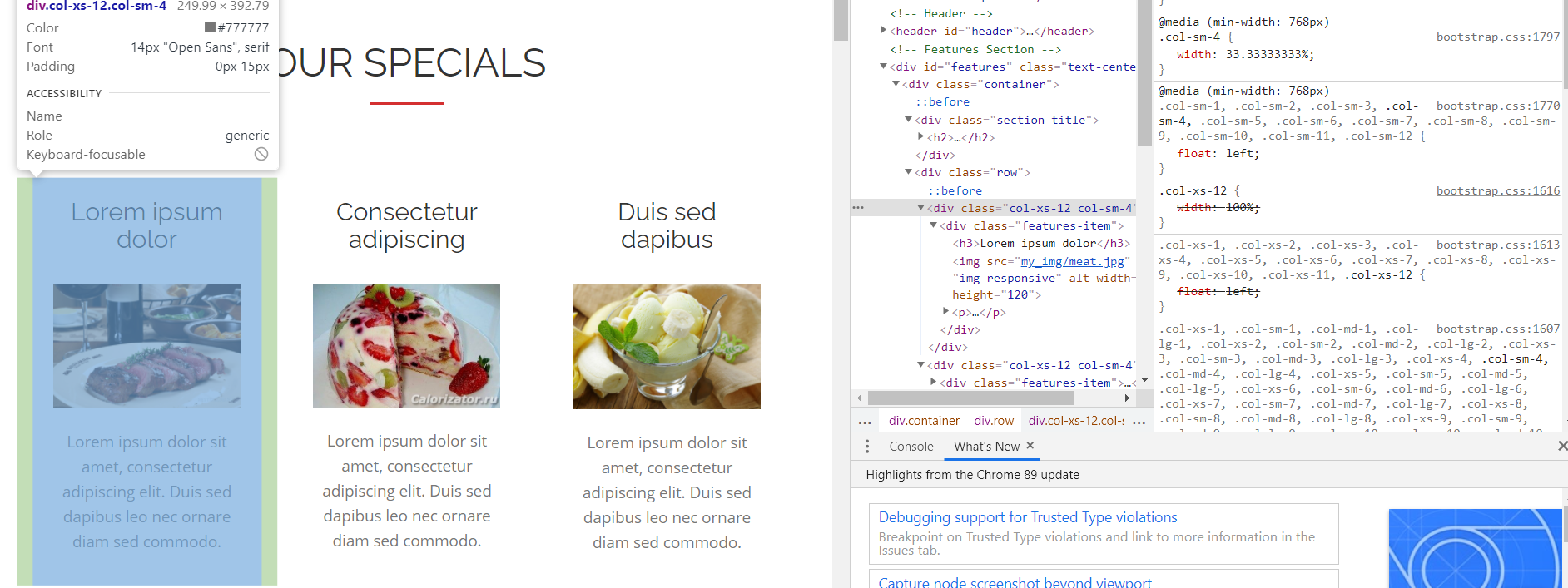
**Block** – блочный бокс. Занимает всю длину и все высоту родительского элемента. Можно задать ширину и высоту.



**Inline-block –** блочно-строчный бокс. Совмещает особенности двух типов. Элементы стают в одну строку, но могут иметь отступы. Например, в шапке можно задать ссылка **inline-block** и отступы по сторонам. **Пример:**

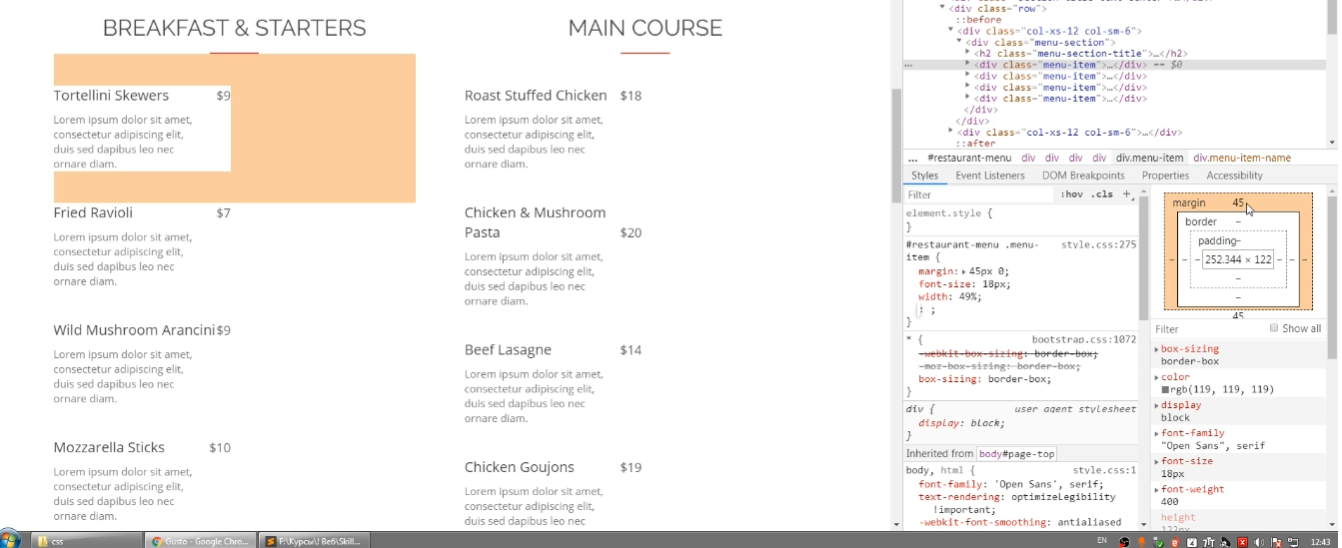


**<div>** мы используем для верстки блоков. Например на сайте ресторана существует множество **div,** через них и оформлена верстка. **Пример:**

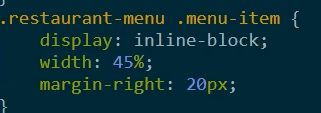


У этих трех картинок блочно задана ширина 33.3, чтобы они одинаково занимали всю площадь страницы. На данном сайте используется **CSS-фреймворк Bootstrap.** Это можно увидеть на правой части скриншота. Это специальная система из готовых **CSS** правил, которая позволяет быстро создать готовый сайт.

В инструменте разработчика можно не только изменить ширину, но и видеть различные отступы.



Удобно сначала работать в инструменте, а потом уже переносить изменения в код.

 Напоминание, эти изменения коснуться классов **menu-item,** которые будут расположены внутри **restaurant-menu.**

**Верстка под экран.**

Важно, чтобы сайт хорошо смотрелся на любом экране, от маленького смартфона, до большого телевизора.

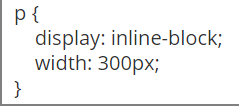
Верстка, которая занимает только одну ширину называется **фиксированной**. Ее сейчас редко используют, так не надо. Есть даже понятие **pixel perfect,** когда верстка равна макету в точности до пикселя.

Верстку, которая подстраивается под ширину бывает двух видов. **Резиновая** и **адаптивная.**

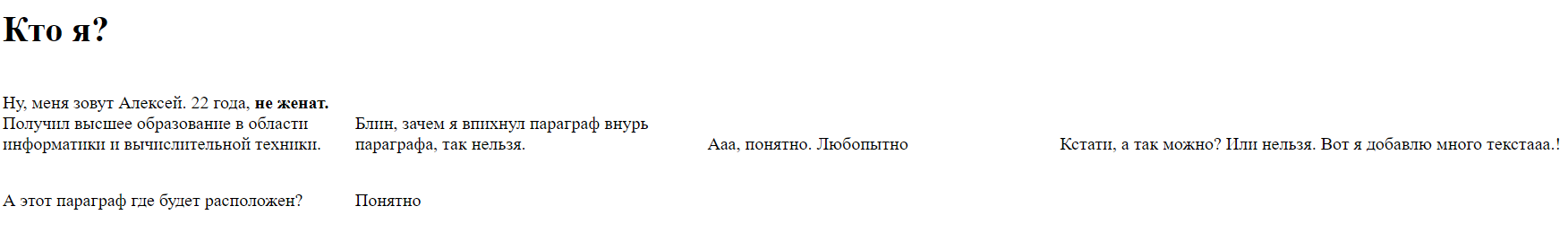
Резиновая меняет ширину элементов, при этом их расположение не меняется. А адаптивная раскладка подстраивается под ширину. Что-то может меняться местами, превращаться в другие (меню в кнопку) элементы или исчезать.

Еще раз напомню. **Не стоит фиксировать высоту.**

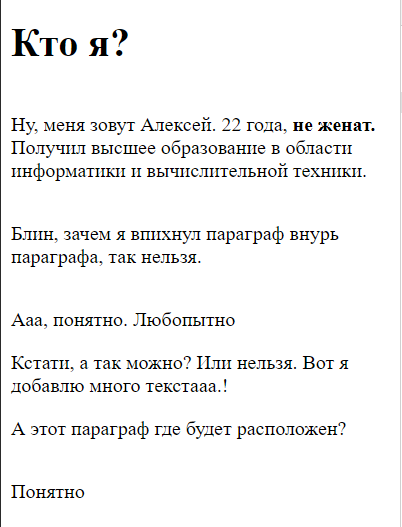
Допустим параграфам будет добавлен стиль, который сделает отображение блочно-строчным с определенной шириной.



В таком случае на для параграфов добавится хоть какая-то адаптивность. Так еще и они сами станут вот в такой ряд.



Но если сайт ужать, то они адаптивно станут друг-под друга.



**Важно:** Ширину текста можно регулировать особой единицей измерения – **ch**, то есть **char** – знаков. Зависит от размера шрифта, но в общем варьируется от 60 до 100.

Все выше сказанное относится к **сплошному тексту.**

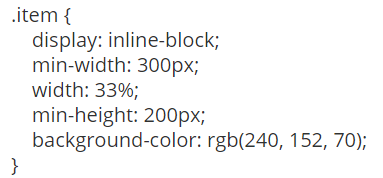
**Информацию можно верстать по-разному.**  Например, список мебели можно сверстать не таблицей а через **div** и **span.**

**Вьюпорт (viewport**) — это видимая область окна браузера, содержимого сайта.

**Блочные боксы.**

Помимо того, что им можно задать ширину и высоту, им также можно задать пороги, т.е. минимум и максимум: **min-height, max-height, min-width, max-width.**

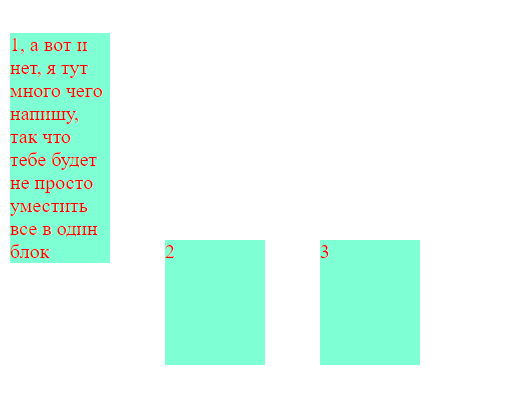
**Пример:**





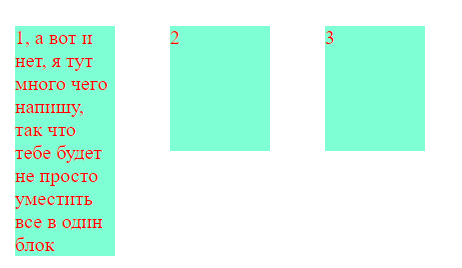
В нем ширина блока равна трети от родителя. Минимальная высота, 200 пикселей, а минимальная ширина 300 пикселей. При малом содержимом блок будет высотой в 200. Ну это очевидно.

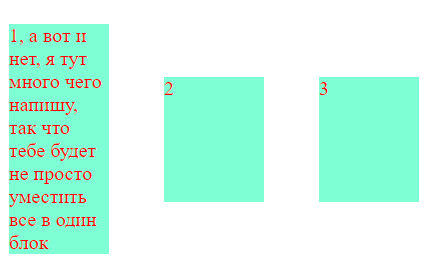
Но если один из блоков будет переполнен, то другие **съедут**

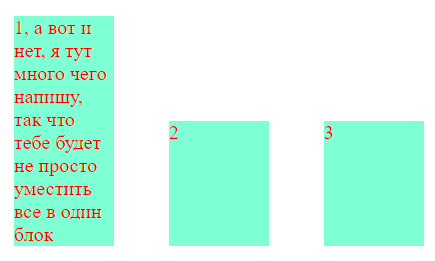


Это произошло, из-за того, что они блочно-строчные и выравниваются по строке. Чтобы это поправить, есть такое свойство:

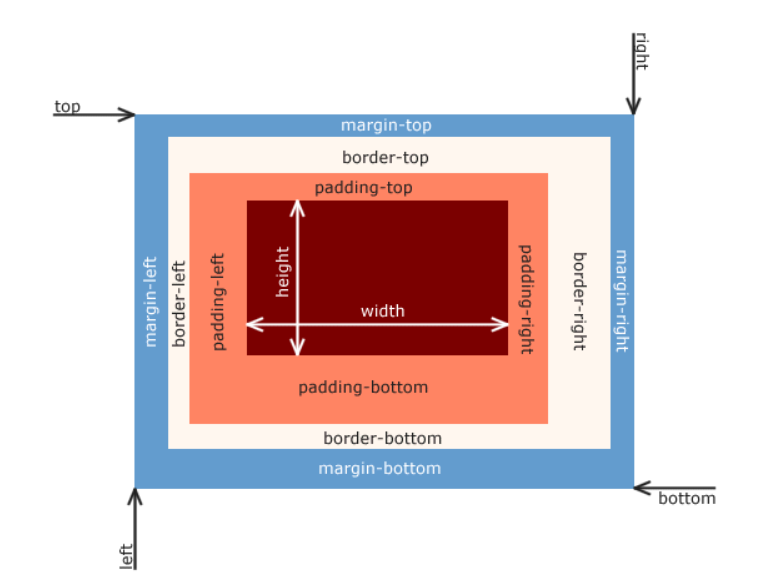
**vertical-align(**перевод вертикальные выравнивание**).** Одно из его значений **bottom(**перевод нижний**)**. Вот примеры:

**Top**

**middle**

**bottom.**

**Пришло время подробнее разобрать блочную модель.**



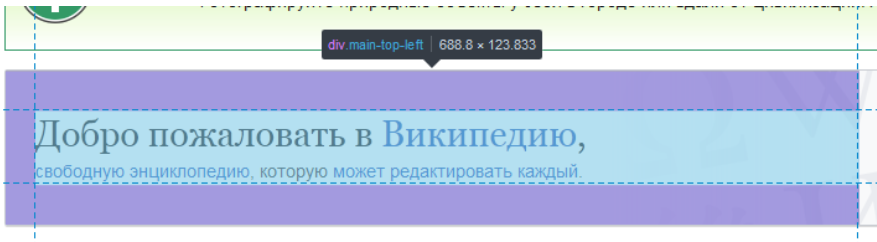
Помимо содержимого со своей шириной и высотой, у блочного бокса могут быть **границы**  и **отступы.**

Отступы бывают двух видов. **Внутренние – padding.** Между границей и содержимым.

И **внешние – margin.** Вокруг границы. Отодвигает элементы друг от друга и от краев страницы.

**Border –** граница. Проходит по периметру элемента.

**Классический пример внутреннего отступа.**



Блок с границей и внутренним отступ. **Такой сложнее читать.**

**Универсальные свойства:**

**Top –** верх

**Right –** правая сторона

**Bottom –**низ

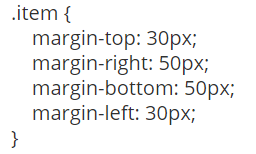
**Left**- левая сторона

Именно в таком порядке. Можно запомнить словом **T R ou B L e.**

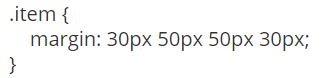
Или просто по часовой стрелке от 12, 3, 6, 9.

**Зачем это запоминать?**

**А дело в том, что margin или padding можно записывать как по порядку каждый:**



**Так и разом:**



**Кстати, если отступы по высоте или ширине не отличаются. То можно ограничится двумя параметрами:**



Добавив отступы, мы увеличили размер самого элемента. Даже если мы указали 33% от родительского элемента, в строке может поместится два, так-как с отступами элементы занимают больше места.

Если одному элементу надо присвоить несколько классов, то их нужно указывать как атрибут через пробел.

R и M названия классов.

Ширину блоков можно регулировать, по умолчанию это только **ширина содержимого.**

Но часто удобнее считать шириной содержимого еще и **внутренний отступ с границей.**

Для этого есть свойство **box-sizing.** По умолчанию оно: **content-box.** Ширина только контент. Но можно изменить на **border-box.** То есть ширина до границ.

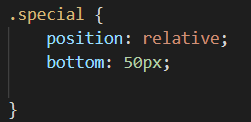
**Позиционирование и наследование.**

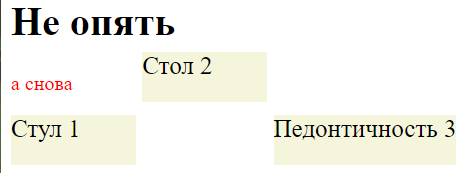
Еще один способ расположение элемента на странице — это **позиционирование.** Свойство **position.** По умолчанию значение **static.** В таком случае, оно располагается в потоке, т.е. в соответствии с порядком в html коде, значениями свойств **display** и **box Model.** Но это можно изменить.

Позиционирование следует применять к одиночным элементам, а не для раскладки всей страницы.

**Position: relative –** позволяется переместить элемент относительно изначального места расположения.

После того, как свойство приобрело это значение, его можно сместить относительно начального через отдельные свойства: **top, right, bottom, left**. **Пример:**

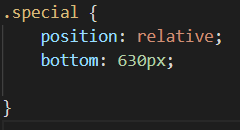


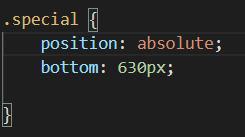


**Абсолютное позиционирование – position: absolute;**

Слегка сложнее.

Сначала покажу разницу между **relative** и **absolute**

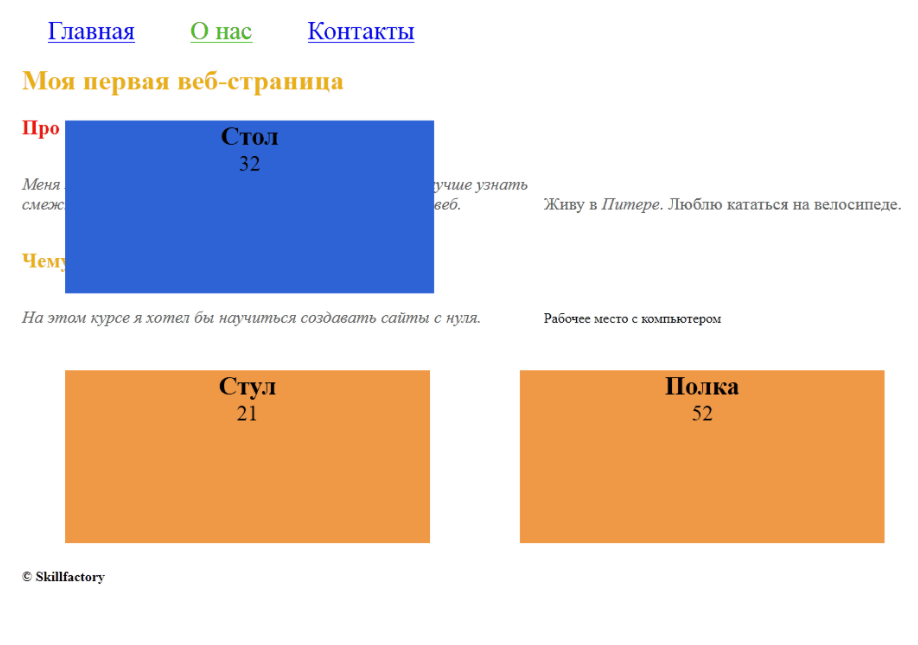




**Как можно заметить при absolute положение объекта заменяется, последующим. Кроме того, сам absolute объект будет накладываться поверх других элементов.**

Помимо ширины и высоты есть и третье измерение. Элементы с абсолютным и не только позиционирование могут наслаиваться друг на друга. Принцип их накладывания описан на этом сайте: <https://developer.mozilla.org/ru/docs/Web/CSS/CSS_Positioning/Understanding_z_index/The_stacking_context>

Пример накладывания при position absolute

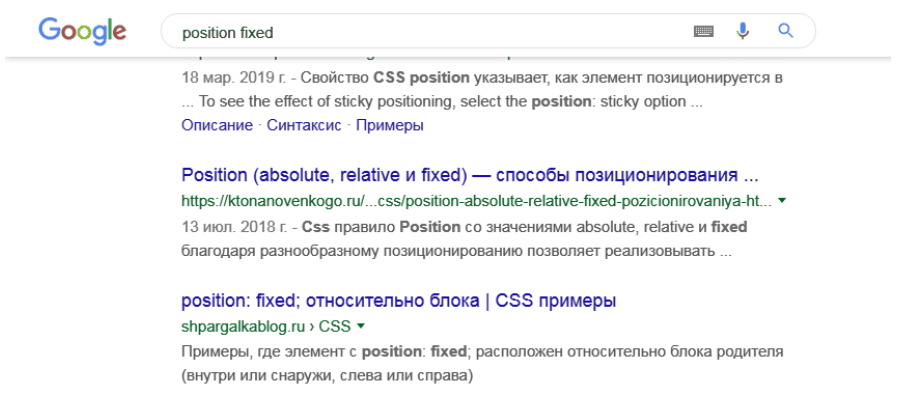


Третье значение **position: fixed**. В отличии от absolute вместо краев экрана, задает положение относительно краев **вьюпорта.**

**Вьюпорт (viewport**) — видимая на экране область документа.

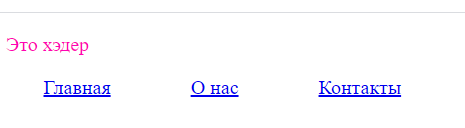
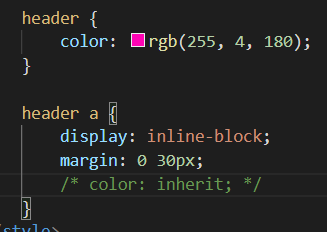
Больше про позиции: <https://developer.mozilla.org/ru/docs/Web/CSS/position>

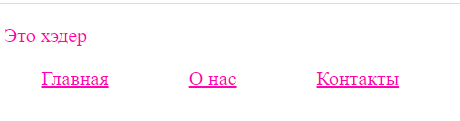
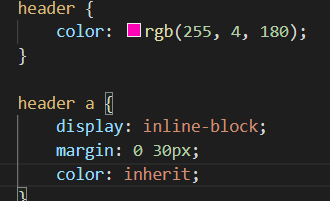
Его используют, чтобы прикрепить вернее меню к краю экрана. Пример:



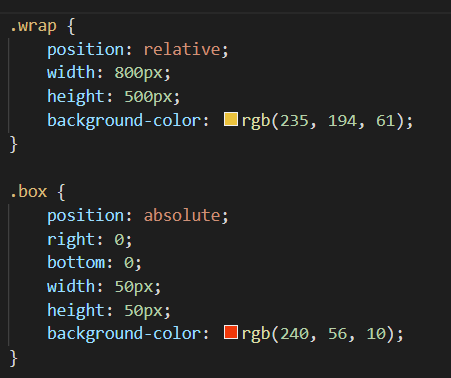
Некоторые значения свойств наследуются от родителей. В основном это свойства текста. Например, если **body** задать свойство для текста, то все параграфы, которые находятся в **body** и не имеют свойств, будут наследовать этот параметр.

Другие свойства не наследуются, например, отступы. Наследованием можно управлять. Например цвет для ссылок сам по себе не наследуется, но с помощью параметра **inherit,** можно разрешить наследование. Если же наоборот, не нужно наследовать, то есть обратное свойство: **initial.**





**Задание А.12.6**



Wrap это поле, где находится квадрат. Задаем ему позиционирование, относительно родителя. А затем box – квадрату задаем позиционирование absolute, и ставим его в нулевые положения справа снизу. Когда мы использует top left bottom right мы указываем на сколько двинутся **от выбранной стороны.**



**Вебинары по позиционированию.**

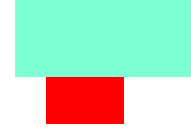
Позиционирование нельзя использовать для раскладки. Это неправильно. Для этого будут flex-box’s и Grid.

**Relative** нельзя. В нем объект сдвигается относительно родителя. И важно помнить, что top, right, bottom left это стороны откуда будет проходить сдвиг.

**Absolute** нельзя, сломается если есть хоть какая-то адаптивность. Но есть задачи, где без него не обойтись. Если ничего не делать, то он будет работать от краев страницы, то есть. Задав bottom 10px он будет в самом низу экрана. Чтобы это исправить и позиционирования изменялось относительно краев родителя, то в родителе должно быть установлено позиционирование отличное от **Static,** т.е любое, кроме него, но как правило используют **relative,** потому что он ни на что не повлияет**.**

**Fixed** похож на absolute. он тоже изымается из потока, но прижимается уже к краю вьюпорта, видимой части экрана.

Сменим тему. Есть такое свойство, **Z-index**. Это свойство параметр наслаивания. Без дробных чисел. Если у элемента index больше (по умолчанию у всех ноль), то, оно сверху и наоборот. Не надо делать гигантские z-index,

У красного z-index -1. Он под голубым.

Чтобы вместо квадрата сделать круг: border-radius: 5px;

**Sticky** похож на **fixed,** но работает конкретно внутри родителя.

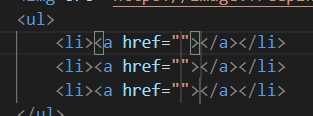
Второй вебинар.

Когда подключен emmet, расширение для быстрого ввода, писать очень просто. Можно написать link, а затем нажать **tab** и она отработает.

Написать doc нажать **tab** и тоже будет целая структура и тд.

**Ctrl enter –** попасть на следующую строку из любого места.

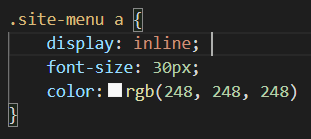
Чтобы быстро создать список пишем такую структуру: **ul>li\*3** Упорядоченный список, внутри которого три элемента. Подобных формул не мало. Если добавить в эту формулу в конце >a, то внутри каждого элемента списка будут ссылки



pexels.com сайт с бесплатными картинками.

В стилях тоже имеется есть быстрый ввод. bc + tab = background-color:

Не получилось выставить в линию ссылки таким образом.



А дело в том, что они находятся внутри **li**, у которого по умолчанию display: list-item. Так что стильно нужно задавать именно ему.

У не упорядоченных списков есть точка. От нее можно избавится через такое свойство:

**list-style: none** 

wrapper – переводится как обертка, так иногда называют классы.

Документация.

У языка **css** свои спецификации, иногда нужно с ней сверяться, так как css сложнее html.

Рекомендуют эту: <https://www.w3.org/TR/CSS/>

Спецификации состоят из уровней и модулей, постоянно обновляются. Некоторые элементы нельзя стилизовать по соображениям безопасности. Например, нельзя изменить размер шрифта у посещенных ссылок. Цвет можно, но размер нельзя.

hover: свойство при наведении

visited: свойство после посещения

<https://developer.mozilla.org/> тут можно искать спецификации о стилях

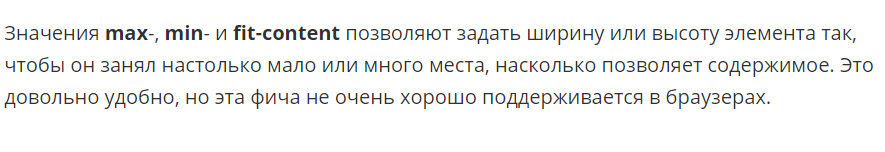
Старые версии браузеров могут не поддерживать новые функции CSS. Да и разные браузеры могут по-разному рассматривать некоторые элементы их стили.

Что чем поддерживается на каких сайтах можно посмотреть тут: <https://caniuse.com/>

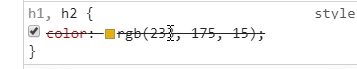
(зеленый цвет), частично (желтый) или не поддерживают (красный).

Пример





Если мы видим в CSS зачеркнутое свойство



То это означает, что оно по какой-то причине не работает. К примеру, оно может быть отключено, потому что его заменяет другое свойство, написанное ниже. Это кстати называется Каскадность, буква Cи в названии CSS.

Стили бывают, пользовательские, ваши и сайта. Приоритет такой же, как порядок в котором они написаны.

Но стили можно писать по-разному. Инлайного, прямо внутри строки, внутри head через атрибут style, и отдельным файлом через <link>

Можно использовать сразу все три способа, и они могут пересекаться, но у них есть разный приоритет.

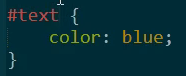
Порядок такой:



Но с отдельным файлом работать проще, поэтому именно его рекомендуют.

Правила с селектором класса имеют больший приоритет, чем общие правила. Один из примеров, у классов специфичность выше чем у селекторов по элементам.

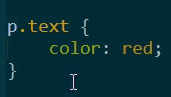
По мимо классов есть еще селекторы id  в стилях они выглядят так



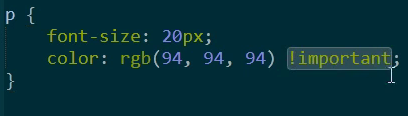
У него специфичность даже выше чем у классов, так что он его переиграет. Но его для стилизации не рекомендуют, потому что его можно применить только для одного элемента на странице. Для других id должен быть с другим названием. Могут возникнуть проблемы, не стоит.

Обычно используют только классы.

У такого стиля будет специфичность больше



Также есть возможность сообщить свойству, его приоритет перед другими.



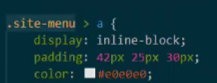
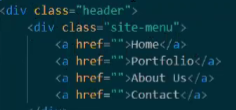
Но так делать не рекомендуют.

**Вебинар – стили веб страниц**. Параллельно исправлю ошибки в своей практике.

Прокрутил в голове, inline только для текста, block это блоки и им можно задавать отступы, но они идут сверху вниз, inline-block это блоки слева на право.

забыл про такое важное свойство, как **vertical-align.**

Когда используется подобный знак **>**

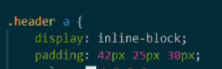
Его применяют, если он должен применится к непосредственным потомкам класса.

Тут видно, что **<a>** это именно потомки класса .**site-menu.**

Если бы вместо **.site-menu** использовался бы класс .**header**,

 то, стиль бы не сработал, потому что непосредственным потомком **header** является блок **<div>,** а ссылки лежат уже внутри этого блока.

Но, если убрать знак **>,** но оставить **header,**

 То стиль сработает.

Свойство **border** – создает рамку вокруг элемента, обычно его задает таким образом:

 толщена, тип рамки, цвет.

**Вебинар «Разбор сложных мест в модуле»**

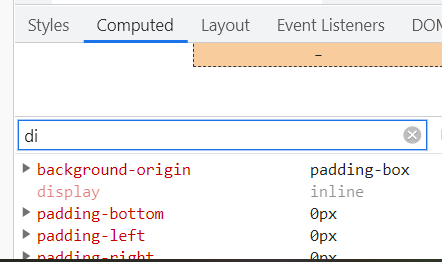
Когда используется emmet, и через него создается сслылка, link в ней может появится такая строка:

 она уже устарела, ее можно удалить.

Как уже известно у объектов есть свою специфичность, у классов элементов id и так далее, иногда в этом можно запутаться, но всегда можно воспользоваться калькулятором. <https://specificity.keegan.st/>

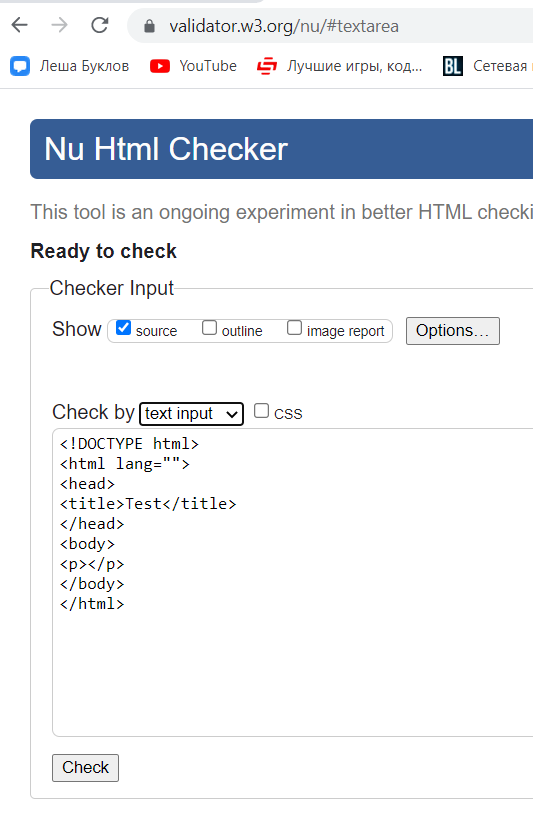
У некоторых элементов есть css по умолчанию. Например, у заголовка **<h1>** есть отступы по умолчанию, размер шрифта и цвет.

В инструментах разработчика некоторые свойство по умолчанию лежат в разделе Computed, в поле filter можно вбить интересующее свойство.



Вспоминаем про валидатор, вот он. <https://validator.w3.org/nu/#textarea>

Если нажать галочку можно вставить и css



Итого, не использовать позиционирование при раскладке, не ограничивать высоту. В итоге inline-block устарел, flex-box и greed его полностью заменят.

**Вебинар «Почему важно уметь искать нужную информацию?»**

Ссылка на презентацию <https://vallek.github.io/web-pres/search.html>

Гуглить не стыдно, нельзя все держать в голове, да и знания постоянно обновляются.

Вся информация до 2014 скорее всего устарела. Финальная версия flexbox была закончена в 2014, а гриды еще позднее. Короче сверять информацию полезно.

Ненадежные сайта: htmlbook, w3schools webref, garu.site, ответы mail.

Это не значит, что их использовать нельзя, но информация там может быть не точна или устаревшая. htmlbook – устаревший сайт.

На вики полезно смотреть факты, даты.

На wiki есть раздел обсуждения.

В соц сетях тоже можно искать инфу, например, есть неплохие сообщества вк.

HABR – коллективный блок, в котором есть много статей.

Reddit.

Даджесты. Например на Хабре. Даджесты, подборки с ссылками.

stackoverflow. – сайт где можно задавать вопросы и искать ответы.

хорошие сайты.

html5book.ru

css-live.ru

smashingmagazine.com

css-tricks.com

Надежные сайты:

developers.google.com

web.dev

developer.mozilla.org

Если в поисковике ввести запрос “в кавычках”, то эта информация будет искаться конкретно буква, в букву. А не общее.

Если на сайте нет встроенного поиска, то можно что-то найти в гугл через такой запрос:

“запрос site:адрес сайта”.

! + enter = заполненная html структура.