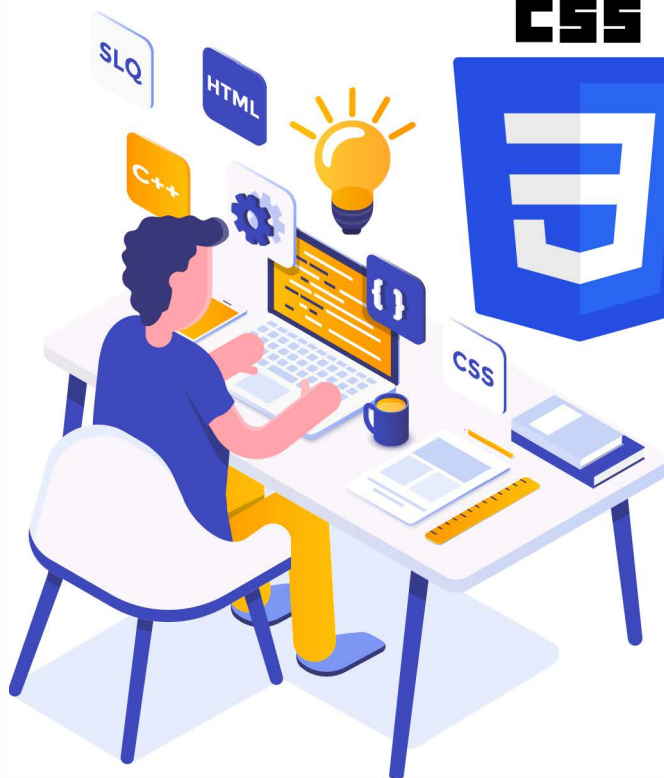


Создание web-страниц с использованием HTML5 и CSS3

HTML

CSS



Урок 5

Блочная модель элементов. Псевдоклассы

Содержание

Расчет реальной ширины элемента.	
Свойство box-sizing	4
Свойство display: inline-block, inline и block.	
Разница между ними	15
Строчные элементы	17
Строчно-блочные элементы	19
Использование псевдоэлементов	22
Псевдоэлемент ::first-line	23
Псевдоэлемент ::first-letter	25
Псевдоэлемент ::selection	27
Псевдоклассы для элементов	29
Псевдокласс :nth-last-child(n)	34
Псевдокласс :only-child	35
Псевдоклассы типа :nth-of-type	38

Использование псевдоклассов при верстке блока страницы	45
Создание колонок с помощью строчно-блочных элементов	70
Использование строчно-блочных элементов для верстки страницы.....	88
Домашнее задание.....	105
Задание 1	105
Задание 2	106

Материалы урока прикреплены к данному PDF-файлу. Для доступа к материалам, урок необходимо открыть в программе Adobe Acrobat Reader.

Расчет реальной ширины элемента. Свойство `box-sizing`

В прошлом уроке мы рассмотрели варианты назначения ширины для блочных элементов. Однако при назначении для элемента не только свойства `width`, но и внутренних отступов (`padding`), а также рамок (`border`) вы обязательно столкнетесь с тем, что реальная ширина элемента больше заданного вами размера для свойства `width`.

Дело в том, что браузер определяет ширину элемента, как комплекс свойств, в котором `width` — это ширина именно контента, к которой добавляются размеры внутренних отступов, границ и внешних отступов. Поэтому при расположении 2х и более элементов рядом в одной линии, необходимо учесть все эти размеры.

Т.е. элемент с такими свойствами:

```
.elem {  
  width: 500px;  
  padding: 15px;  
  border: 2px solid #ccc;  
  margin: 20px;  
}
```

будет в браузере иметь расчетную ширину в 574px ($500\text{px} + 2 \cdot 15\text{px} + 2 \cdot 2\text{px} + 2 \cdot 20\text{px}$). Итого мы имеем 574px вместо 500px, указанных в свойстве `width`.

Можно сказать, что реальная ширина элемента в браузере рассчитывается по такой формуле:


```
margin-left + border-left + padding-left + width +
padding-right + border-right + margin-right
```

Та же ситуация складывается с высотой элемента. Она определяется по формуле:

```
margin-top + border-top + padding-top + height +
padding-bottom + border-bottom + margin-bottom
```

Высоту элемента приходится высчитывать реже, чем ширину, т.к. в большинстве случаев она определяется контентом и имеет значение **auto**, а распределять колонки по ширине — это постоянная задача верстальщика, т.к. на данный момент верстка колонками — это тренд, особенно для посадочных страниц (*Landing Page*).

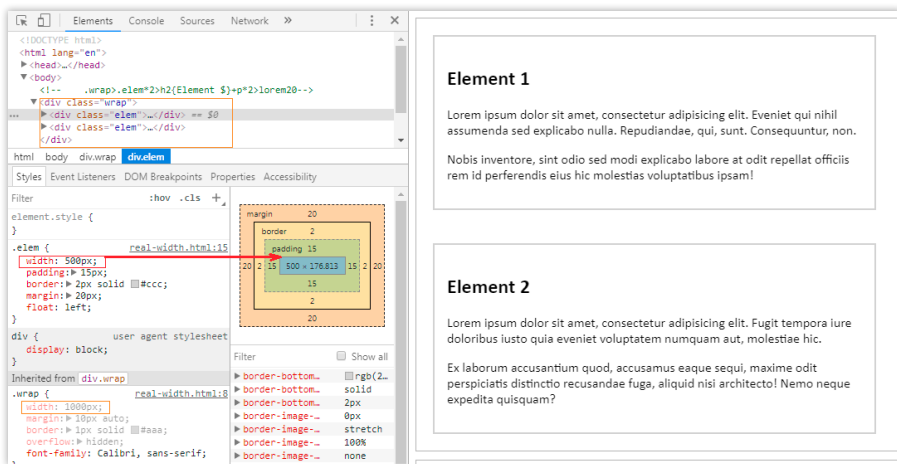


Иллюстрация 1

На скриншоте (ил. 1) хорошо видно, что 2 плавающих элемента с классом *elem* и шириной в 500px не помещаются

в контейнер класса *wrap* с шириной в 1000px, хотя по правилам математики ($500\text{px} \times 2 = 1000\text{px}$) должны. Происходит это из-за увеличения расчетной ширины элемента в браузере, за счет добавления отступов и рамок.

Посмотреть пример можно в файле *real-width.html* в папке *example* этого урока.

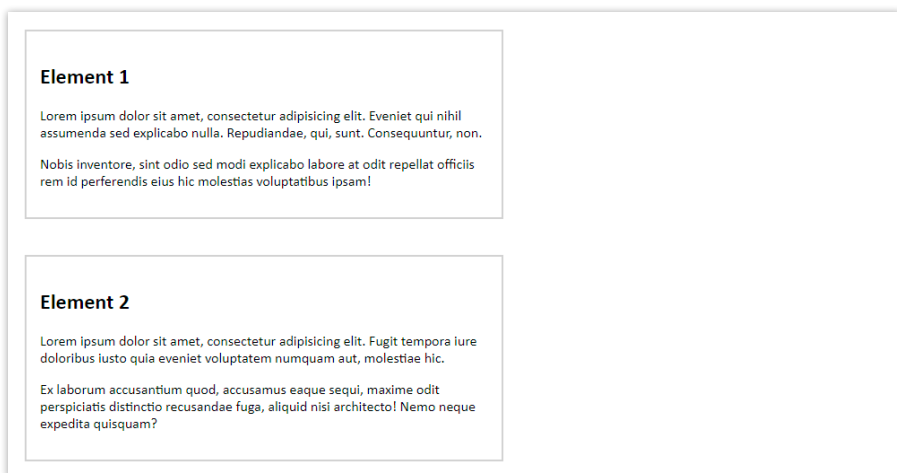


Иллюстрация 2

Что же делать в этом случае? Есть 3 варианта выхода из ситуации: первый, основанный на расчете ширины элемента с учетом отступов и рамок, второй, который часто использовался ранее, подразумевающий наличие вложенного элемента, и третий, современный — с использованием свойства **box-sizing**.

Поддержка этого свойства появилась еще с 2006 года (Mozilla Firefox v.2) и с 2010 в Chrome 4, но тогда оно использовалось с вендорными префиксами (**-moz-**, **-webkit-**), т.к. в стандартном виде еще не получило статус рекомендованного организацией W3C (ил. 3).

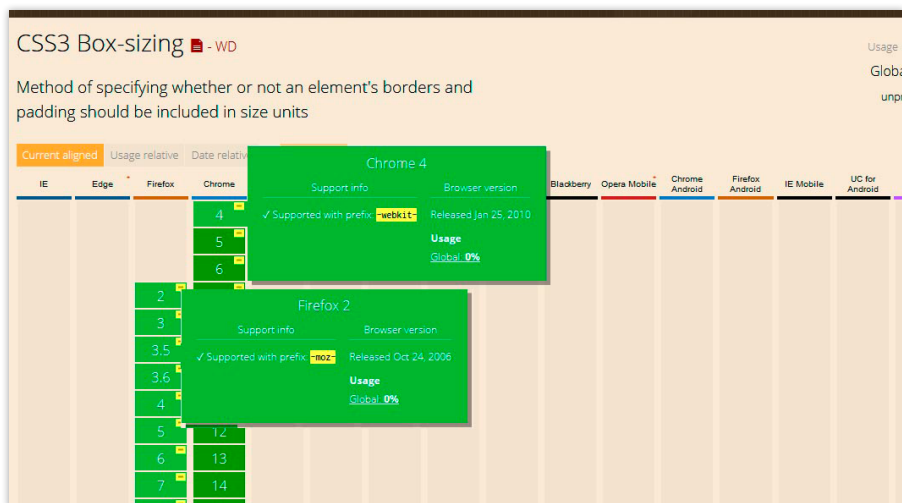


Иллюстрация 3

На данный момент все браузеры, включая Internet Explorer с 8-й версии, поддерживают это свойство без вендорных префиксов. Данные о поддержке этого свойства вы можете на сайте: caniuse.com (ил. 4).

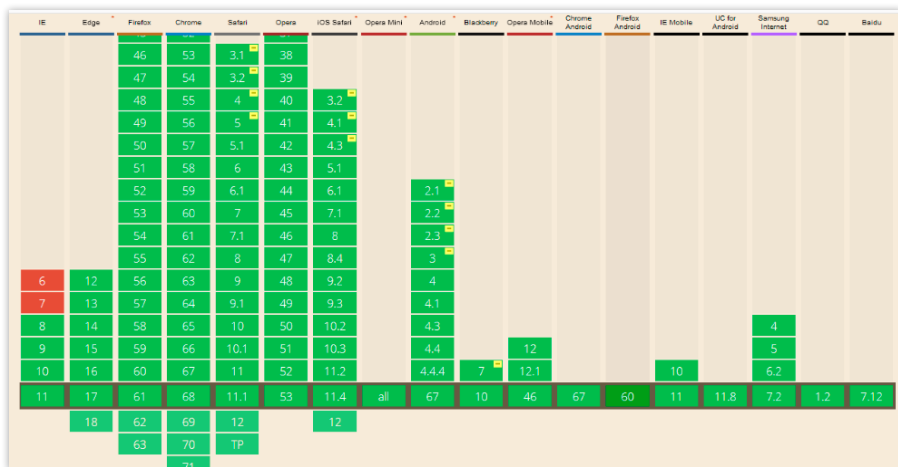


Иллюстрация 4

Первый способ отталкивается от нужной ширины элемента, но в свойстве **width** нужно указать значение без внутренних и внешних отступов и без рамок. Т.е. придется считать значения, но не прибавляя, а отнимая величины указанных свойств: $500\text{px} - 2 \cdot 15\text{px}$ (**padding**) $- 2 \cdot 2\text{px}$ (**border**) $- 2 \cdot 20\text{px}$ (**margin**) $= 426\text{px}$

```
.elem-real {
  width: 426px;
  padding: 15px;
  border: 2px solid #ccc;
  margin: 20px;
  float: left;
}
```

В Инспекторе свойств можно посмотреть, как распределяются все заданные нами свойства box-модели (ил. 5).

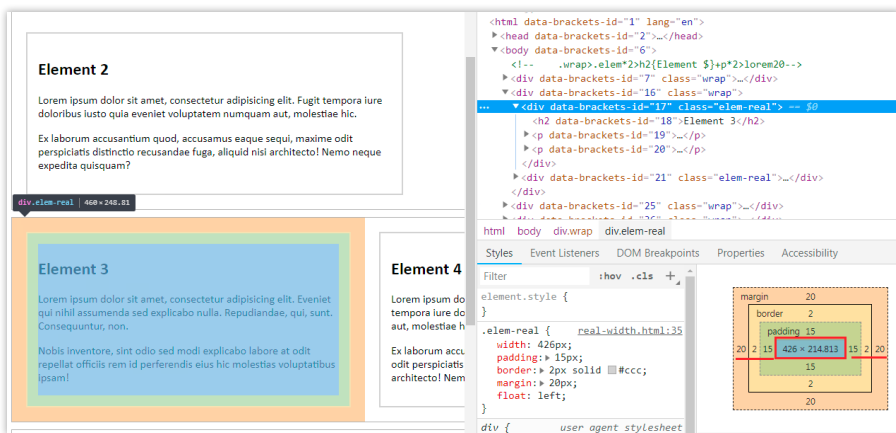


Иллюстрация 5

Второй способ — добавить в html-разметку 2 элемента с разными классами вместо одного:

```

<div class="elem-parent">
  <div class="inner-elem">
    <h2>Element 1 inner</h2>
    <p>Lorem ipsum dolor sit ...</p>
    <p>Nobis inventore, sint...</p>
  </div>
</div>

```

А в стилях распределим свойства, которые записывали для класса *elem* между двумя классами таким образом:

```

.elem-parent{
  width: 500px;
  float: left;
}

.inner-elem {
  padding: 15px;
  border: 2px solid #ccc;
  margin: 20px;
}

```

В результате получаем 2 расположенных рядом элемента (ил. 6).

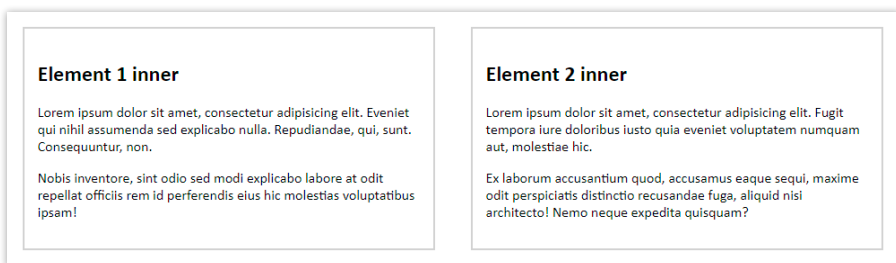


Иллюстрация 6

В инспекторе свойств видно, что размеры вложенного элемента с классом *inner-elem* совпадают с теми, которые мы рассчитывали в первом способе, хотя свойство **width** для него не задавалось.

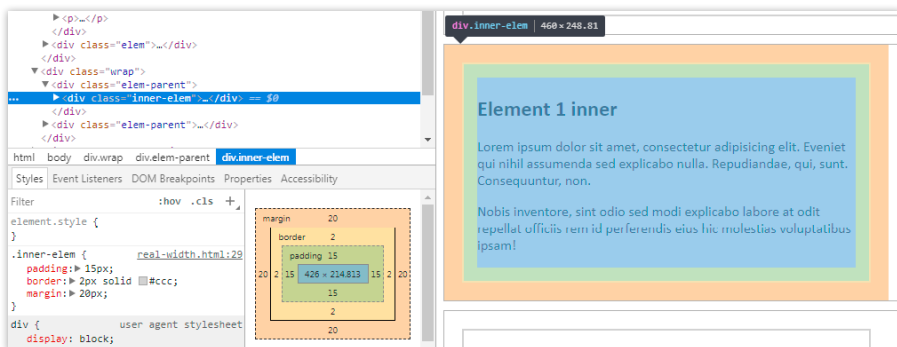


Иллюстрация 7

Поскольку добавление дополнительного класса не является самым лучшим решением, рассмотрим третий способ. Он основан на применении свойства **box-sizing**, которое может иметь следующие значения:

```
box-sizing: content-box | border-box | initial | inherit
```

По умолчанию свойство **box-sizing** имеет значение **content-box**, т.е. свойства **width** и **height** задают размеры содержимого и не включают значения отступов **padding** и **margin** и рамок **border**. Именно этот вариант приводит к увеличению реальной ширины элемента в браузере при добавлении отступов и границ.

Значение **border-box** подразумевает, что в состав свойств **width** и **height** включаются значения *padding* и *border*, но не *margin*. То есть ширина или высота элемента задается

с учетом внутренних отступов и рамок. Внешние отступы не учитываются, т.к. их не всегда нужно задавать с обеих сторон, иногда достаточно только внешнего отступа только справа или снизу.

С учетом этого свойства со значением **border-box** правила для наших элементов становятся такими:

```
.elem-sizing {
  box-sizing: border-box;
  width: 460px;
  padding: 15px;
  border: 2px solid #ccc;
  margin: 20px;
  float: left;
}
```

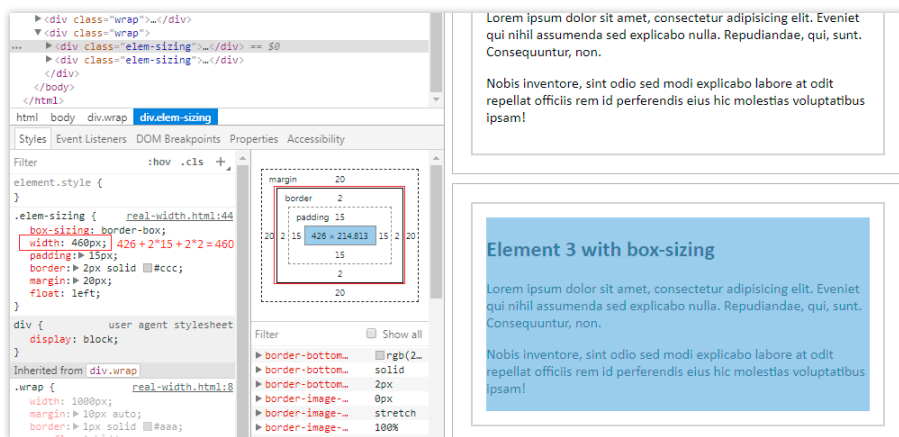


Иллюстрация 8

Ширина элемента составляет 460px, т.к. необходимо вычесть значения 2-х внешних отступов по 20px из 500px. На скриншоте (ил. 8) видно, что 460px складываются из 426px контента, 30px (15px*2) внутренних отступов

(padding) и 4px (2px*2) рамок (border). То есть мы опять пришли к тому значению ширины контента, которое рассчитали в первом способе.

В результате получим 2 размещенных рядом блока без особых расчетов и дополнительных элементов (ил. 9).

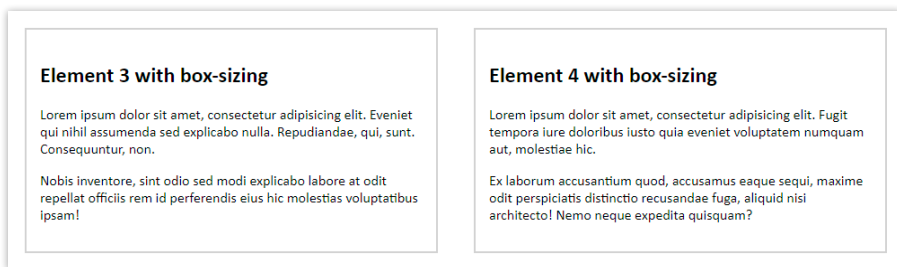


Иллюстрация 9

Хотелось бы отметить, что именно последний способ является сейчас наиболее распространенным, т.к. позволяет задать ширину элемента, не задумываясь о размерах его внутренних отступов или рамок и без дополнительного элемента в разметке, и в этом состоит его отличие от первого способа, в котором ширина связана с учетом для свойства **width** значений **padding** и **border**, которые могут меняться и, соответственно, ширину все время необходимо заново вычислять, например, для разных разрешений экранов, и от второго способа, в котором вложенный элемент не нужен с точки зрения логики (или семантики) разметки. Возможно, вы сейчас этого еще не осознаете, но это отличный выход при изменении размеров внутренних отступов и рамок: вам не нужно пересчитывать размеры элемента. И что особенно ценно, при назначении ширины элемента в %, не имеет значения, какими будут его отступы — ширина контента обязательно

будет рассчитана с учетом всех величин — и это работа браузера, а не верстальщика.

На данный момент это настолько широко используемое свойство, что с него обычно начинается верстка любой страницы. Чаще всего его задают сразу для всех элементов, используя универсальный селектор:

```
*{ box-sizing: border-box; }
```

Кроме того, желательно указывать это свойство для псевдоэлементов `::before` и `::after`, т.к. они очень часто используются для формирования различных эффектов на странице, в том числе и анимационных:

```
*, *::before, *::after {  
    box-sizing: border-box;  
}
```

Вы можете найти еще такое написание свойства `box-sizing`:

```
*, *::before, *::after {  
    -webkit-box-sizing: border-box;  
    -moz-box-sizing: border-box;  
    box-sizing: border-box;  
}
```

Такая запись подразумевает, что сначала указываются вендорные префиксы для браузеров, которые не поддерживают стандартное свойство, а затем уже само это свойство. На данный момент в этом нет необходимости, т.к. по данным сайта caniuse.com глобальное использование

версий браузеров, для которых нужны префиксы, составляет почти 0%.



Иллюстрация 10

Ссылки по теме:

1. https://www.w3schools.com/css/css_boxmodel.asp.
2. https://www.w3schools.com/css/css3_box-sizing.asp.
3. <https://developer.mozilla.org/en-US/docs/Web/CSS/box-sizing>.
4. <https://css-tricks.com/box-sizing/>.
5. <https://webref.ru/css/box-sizing>.

Свойство display: inline-block, inline и block. Разница между ними

Одним из важных свойств box-модели является свойство **display**, которое отвечает за отображение элементов на странице и имеет следующие значения:

Значение	Описание
block	блочные элементы
inline	строчные элементы
inline-block	строчно-блочные элементы
none	элемент не отображается
table, inline-table	таблица
table-row	строка таблицы
table-cell	ячейка таблицы
table-column	столбец таблицы, созданный элементом <col>
table-column-group	один или несколько столбцов таблицы, созданных элементом <colgroup>
table-header-group	аналог отображения шапки таблицы с помощью элемента <thead>
table-footer-group	аналог отображения нижней части таблицы с помощью элемента <tfoot>
table-row-group	аналог отображения тела таблицы с помощью элемента <tbody>
list-item	элемент списка с маркером
flex, inline-flex	контейнеры для гибких элементов
grid, inline-grid	контейнеры для элементов сетки
run-in	блочные или строчные элементы в зависимости от контекста (неполная поддержка браузерами)

Значение	Описание
inherit	наследует значение свойства от родителя
initial	устанавливает значение свойства по умолчанию

По умолчанию для каждого элемента значение свойства **display** назначается на основе рекомендаций, изложенных в спецификации HTML, и берется из таблицы стилей браузера. Как и любое другое css-свойство, его можно переопределить для каждого конкретного элемента или группы элементов с помощью css-селекторов.

Это свойство имеет ряд значений, которые мы будем рассматривать в соответствующих темах. На прошлом уроке мы подробно останавливались на свойствах box-модели, которая определяется чаще всего для блочных элементов (свойство **display** имеет значение **block**). Для сегодняшнего урока нам будут важны такие значения свойства **display**, как **inline** и **inline-block**. Рассматривать мы их будем применительно к блочной модели элементов, т.к. имеется ряд особенностей, связанных с отображением различными элементами таких свойств, как **width**, **padding**, **margin** в зависимости именно от назначенного им свойства **display**.

Полезные ссылки:

1. <https://helpx.adobe.com/ru/indesign/using/anchored-objects.html>.
2. <https://www.w3.org/TR/css-display-3/>.
3. <https://www.w3.org/wiki/CSS/Properties/display>.
4. https://www.w3schools.com/cssref/pr_class_display.asp.
5. <https://developer.mozilla.org/ru/docs/Web/CSS/display>.
6. <https://learn.javascript.ru/display>.
7. <https://webref.ru/css/display>.

Строчные элементы

Строчные элементы имеют свойство **display: inline**, поэтому занимают ровно столько места, сколько в них находится текста. Если этим элементам задать фоновый цвет, внутренние и внешние отступы и разместить их в блоке с большим количеством текста, то можно увидеть не слишком красивую картину (ил. 11).

inline elements

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Sint quae, error iure. Unde iure, aliquam provident sequi cum debitis, reiciendis nesciunt distinctio ut culpa, quos ulla, eveniet similique minus et recusandae a velit ad sed excepturi deserunt optio! Delectus, quasi.

Earum perspiciatis doloremque fugiat repellendus nam dolorem placeat maiores temporibus voluptate dicta. Quis aperiam nihil illo a debitis nam, facilis, velit sed vero. Facere voluptatum unde debitis laboriosam, odio excepturi consequuntur at nihil optio nemo incidunt vitae, qui iste dolor.

A nisi fugit, consequuntur cum qui ut rerum dolorum pariatur! Sed pariatur, dolor explicabo quo provident minus repudiandae quisquam, modi quaerat sint. Veniam corporis aliquid voluptates animi, molestias earum rerum. Dolorem est laboriosam unde eos amet quia excepturi neque quos!

In et dolorem voluptatibus sit incidunt odio reiciendis, nesciunt deserunt illum, earum tenetur repellat est recusandae rem aliquid quam facillae blanditiis nam ad, nam expedita sequi adipisci similique repellendus. Asperiores recusandae, perspiciatis id deleniti, incidunt deserunt rem blanditiis quae quas.

Nulla illum neque ab nisi error ea, dolorum officiis, labore voluptatibus, doloremque esse quaerat quo quae ad nostrum ipsa modi, eaque facilis quis! Ut porro necessitatibus consectetur, voluptatibus ab laudantium sapiente facilis a, aperiam maiores, nulla quis voluptates, aliquid cupiditate.

Иллюстрация 11

Посмотреть пример можно в файле *inline-and-inline-block-elements.html* в папке *examples* этого урока. CSS-свойства для выделенных элементов таковы:

```
.inline {
    background-color: yellow;
    padding: 10px;
    margin: 10px;
}
```

Из скриншота видно, что фон строчных элементов с классом *inline* накладывается на текст, идущий за ним, перекрывая его и делая его нечитабельным. Избежать этого можно либо уменьшением внутренних отступов (*padding*), либо установив большую высоту строк (свойство *line-height*). Есть еще один способ борьбы — сделать эти элементы строчно-блочными, указав для них свойство *display: inline-block*.

Примечание. В файле примера вы найдете закомментированную строку аббревиатуры *Emmet*, которая позволяет сгенерировать текст примера: `<!-- h1{Inline and inline-block elements}+ h2{inline elements}+ p*5>lorem40^h2{inline-block elements}+ p*5>lorem40 -->`

Чтобы убрать комментарий, установите курсор в любое место строки и нажмите *Ctrl + /*, а затем переместите курсор сразу после цифры 40 (без пробелов!) и нажмите клавишу *Tab*.

Строчно-блочные элементы

Строчно-блочные элементы ведут себя, с одной стороны, как строчные, т.е. занимают ровно столько места, сколько в них находится текста, а с другой стороны — интерпретируют отступы, ширину и выравнивание текста так, как это делают блочные элементы.

Сравните внешний вид текста на скриншоте (ил. 12) с тем, который был ранее. В том же файле *inline-and-inline-block-elements.html* из папки *examples* для элементов, которым задавался ранее класс *inline*, был добавлен класс *inline-block* с таким кодом:

inline-block elements

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Sint quae, error iure. Unde iure, aliquam provident sequi cum debitis reiciendis nesciunt distinctio ut culpa, quos ullam, eveniet similique minus et recusandae a velit ad sed excepturi deserunt optio! Delectus, quasi.

Earum perspiciatis doloremque fugiat repellendus nam dolorem placeat maiores temporibus voluptate dicta. Quis aperiam nihil illo a debitis nam, facilis, velit sed vero. Facere voluptatum unde debitis laboriosam, odio excepturi consequuntur at nihil optio nemo incidunt vitae, qui iste dolor.

A nisi fugit, consequuntur eum qui, ut rerum dolorum pariatur! Sed pariatur, dolor explicabo quo provident minus repudiandae quisquam modi quaerat sint. Veniam corporis aliquid voluptates animi, molestias earum rerum. Dolorem est laboriosam unde eos amet quia excepturi neque quos!

In et dolorem voluptatibus sit incidunt odio reiciendis, nesciunt deserunt illum earum tenetur repellat est recusandae rem aliquid quam facilis blanditiis nam ad

porro expedita sequi adipisci similique repellendus. Asperiores recusandae, perspiciatis id deleniti incidunt deserunt rem blanditiis quae quas.

Nulla illum neque ab nisi error ea, dolorum officiis, labore voluptatibus, doloremque esse quaerat quo quae ad nostrum ipsa modi, eaque facilis quis! Ut porro necessitatibus consectetur, voluptatibus ab laudantium sapiente facilis a, aperiam maiores, nulla quis voluptates, aliquid cupiditate.

Иллюстрация 12

```
.inline-block {  
    background-color: pink;  
    padding: 10px;  
    margin: 10px;  
    display: inline-block;  
}
```

Сразу бросается в глаза, что строчно-блочные элементы перестали накладываться на соседний текст, внешние отступы отодвинули их на определенное расстояние от рядом идущих слов и строк. Однако сложно назвать этот текст красиво отформатированным.

Из вышесказанного можно сделать следующие выводы — если вам нужно выполнить выделение текста в предложении фоновым цветом, можно использовать строчные элементы с небольшими внутренними отступами (**padding**), внешние отступы (**margin**) имеет смысл задавать только слева и справа. Также имеет смысл увеличить высоту строк (css-свойство **line-height**). Например, тот же текст, который мы рассматривали выше, можно отформатировать, используя элементы **span** с классом *inline-formatting*, поместив их в **div** с классом *container*, для которого увеличена высота строк:

```
.inline-formatting {  
    background-color: aquamarine;  
    padding: 3px;  
    margin-left: 2px;  
    margin-right: 2px;  
}  
  
.container {line-height: 170%;}
```


Внешний вид текста сразу поменялся (ил. 13) (файл *in-line-and-inline-block-elements.html*):

Formatting text using inline elements

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Sint quae, error iure. Unde iure, aliquam provident **sequi cum debitis** reiciendis nesciunt distinctio ut culpa, quos ullam, eveniet similique minus et recusandae a velit ad sed excepturi deserunt optio! Delectus, quasi.

Earum perspicatis doloreque fugiat repellendus nam dolorem placeat maiores temporibus voluptate dicta. Quis **aperiam nihil illo a debitis nam**, facilis, velit sed vero. Facere voluptatum unde debitis laboriosam, odio excepturi consequuntur at nihil optio nemo incidunt vitae, qui iste dolor.

A nisi fugit, consequuntur eum qui, ut rerum dolorum pariatur! Sed pariatur, dolor explicabo quo provident **minus repudiandae quisquam** modi quaerat sint. Veniam corporis aliquid voluptates animi, molestias earum rerum. Dolorem est laboriosam unde eos amet quia excepturi neque quos!

In et dolorem voluptatibus sit incidunt odio reiciendis, nesciunt deserunt illum earum tenetur repellat est recusandae rem aliquid quam facilis blanditis nam ad **porro expedita sequi adipisci similique repellendus. Asperiores recusandae, perspicatis id deleniti** incidunt deserunt rem blanditis quae quas.

Nulla illum **neque ab nisi error ea, dolorum officiis, labore voluptatibus**, doloreque esse quaerat quo quae ad nostrum ipsa modi, eaque facilis quis! Ut porro necessitatibus consectetur, voluptatibus ab laudantium sapiente facilis a, aperiam maiores, nulla quis voluptates, aliquid cupiditate.

Иллюстрация 13

Если же целью форматирования является визуальное выделение элемента на фоне других — лучше сделать его строчно-блочным и использовать все свойства блочной модели в любых вариациях.

Использование псевдоэлементов

Мы уже рассматривали псевдоэлементы `::before` и `::after`. Кроме них вы можете воспользоваться такими псевдоэлементами, как `::first-line`, `::first-letter` или `::selection`. Рассмотрим, каким образом можно использовать эти псевдоэлементы.

Напомню, что псевдоэлементы записываются с одним (спецификация CSS2 и CSS2.1) или двумя двоеточиями (спецификация CSS3) перед наименованием. Исключение составляет псевдоэлемент `::selection` — он всегда записывается с двумя двоеточиями.

Псевдоэлемент ::first-line

По названию этого псевдоэлемента можно догадаться, что предназначен он для форматирования первой строки блочных элементов. Для оформления первой строки можно использовать не все CSS-свойства, а только те, что перечислены в списке ниже:

1. Свойства группы font (**font-family**, **font-size**, **font-style**, **font-weight**, **font-variant** и др.).
2. color.
3. Свойства группы background.
4. word-spacing.
5. letter-spacing.
6. text-decoration, text-decoration-color, text-decoration-line, text-decoration-style.
7. vertical-align.
8. text-shadow.
9. text-transform.
10. line-height.
11. clear.

Например:

```
p::first-line {  
    color: red;  
    text-shadow: 1px 1px 1px #333;  
    font-family: "Segoe Script", cursive;  
}
```

::first-line

Lorem ipsum dolor sit amet, consectetur adipisicing elit. Odit aliquam, vel molestiae officia accusamus quis! Quae repellendus quo odio soluta adipisci deserunt iusto maxime, distinctio impedit omnis nesciunt deleniti blanditiis neque ad reprehenderit modi sunt inventore, iste similique. Veniam unde nesciunt consequatur debitis earum impedit!

Haque commodi impedit magnam eos maiores, fuga nostrum corporis quos quaerat! Vitae amet, incidunt quas quia iure mollitia fugiat quidem perferendis obcaecati soluta minus maxime ipsum repellat nobis, libero autem deleniti architecto dolorem officii in qui voluptate iste. Voluptatum praesentium quam mollitia velit eaque laborum.

Dolores omnis temporibus quas explicabo, harum minus expedita enim eaque, magnam ea sunt? Esse, saepe. Error unde, deleniti iusto molestiae totam veniam consectetur dolore molestias, a itaque non quia magni cumque sint, dolores ad debitis at, laborum veritatis aperiam labore suscipit praesentium quidem. Neque, aliquid.

Иллюстрация 14

При использовании других свойств вы можете не увидеть никаких изменений. Например, нельзя отодвинуть текст в первой строке, используя псевдоэлемент, с помощью свойства **text-indent** или добавить отступ с помощью **margin**. Зато можно это сделать для всего элемента.

```
p {
  text-indent: 20px;
  margin-left: 20px;
}
```

::first-line

Lorem ipsum dolor sit amet, consectetur adipisicing elit. Odit aliquam, vel molestiae officia accusamus quis! Quae repellendus quo odio soluta adipisci deserunt iusto maxime, distinctio impedit omnis nesciunt deleniti blanditiis neque ad reprehenderit modi sunt inventore, iste similique. Veniam unde nesciunt consequatur debitis earum impedit!

Haque commodi impedit magnam eos maiores, fuga nostrum corporis quos quaerat! Vitae amet, incidunt quas quia iure mollitia fugiat quidem perferendis obcaecati soluta minus maxime ipsum repellat nobis, libero autem deleniti architecto dolorem officii in qui voluptate iste. Voluptatum praesentium quam mollitia velit eaque laborum.

Иллюстрация 15

Псевдоэлемент ::first-letter

По названию этого псевдоэлемента также можно понять, что его предназначение — оформить первую букву блочного элемента. Если вы когда-нибудь сталкивались с буквицей, то представляете, как можно использовать этот псевдоэлемент. Для него тоже есть ограничения по используемым свойствам. Применять можно такие же CSS-свойства, как и для `::first-line` плюс перечисленные ниже:

1. Свойства группы `margin`.
2. Свойства группы `padding`.
3. Свойства группы `border`.
4. `box-shadow`.
5. `float`.
6. `vertical-align` (только если `float` равен `none`).

Например, можно так оформить первые буквы в абзацах:

`::first-letter`

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Odit aliquam, vel molestiae officia accusamus quis! Quae repellendus quo odio soluta adipisci deserunt iusto maxime, distinctio impedit omnis nesciunt deleniti blanditiis neque ad reprehenderit modi sunt inventore, iste similique. Veniam unde nesciunt consequatur debitis earum impedit!

Itaque commodi impedit magnam eos maiores, fuga nostrum corporis quos quaerat! Vitae amet, incidunt quas quia iure mollitia fugiat quidem perferendis obcaecati soluta minus maxime ipsum repellat nobis, libero autem deleniti architecto dolore officii in qui voluptate iste. Voluptatum praesentium quam mollitia velit eaque laborum.

Dolores omnis temporibus quas explicabo, harum minus expedita enim eaque, magnam ea sunt? Esse, saepe. Error unde, deleniti iusto molestiae totam veniam consectetur dolore molestias, a itaque non quia magni cumque sint, dolores ad debitis at, laborum veritatis aperiam labore suscipit praesentium quidem. Neque, aliquid.

Иллюстрация 16

Код примера:

```
p::first-letter {  
    display: inline-block;  
    color: green;  
    font-size: 2em;  
    font-weight: bold;  
    border: 1px solid #1ebc1e;  
    padding: 2px 8px;  
    margin-right: 10px;  
    margin-top: 7px;  
    float: left;  
}
```

Псевдоэлемент ::selection

Псевдоэлемент **::selection** предназначен для изменения вида текста при выделении. Его можно назначить как для всего документа, так и для определенных элементов. Следует помнить, что с этим псевдоэлементом можно использовать небольшое количество CSS-свойств, а именно:

1. color.
2. background-color.
3. cursor.
4. outline и его варианты.
5. Группа свойств text-decoration.
6. text-shadow.

Для браузера Mozilla Firefox используется нестандартное написание этого псевдоэлемента **::-moz-selection**.

Для всех элементов в документе:

```
::selection {  
    background-color: #fcfc6b;  
}
```

Для абзаца:

```
p::-moz-selection {  
    background-color: #025302;  
    color: #fff;  
}
```

```
p::selection {
    background-color: #025302;
    color: #fff;
}
```

::first-letter

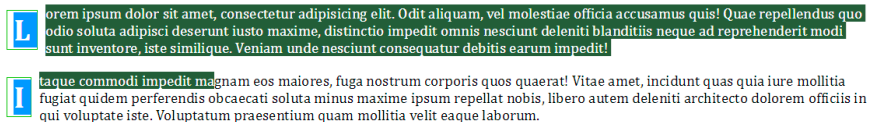


Иллюстрация 17

Код примеров вы можете найти в файле *pseudoelements.html* в папке examples.

Полезные ссылки:

1. <https://developer.mozilla.org/en-US/docs/Web/CSS/::first-letter>.
2. <https://developer.mozilla.org/ru/docs/Web/CSS/::first-letter>.
3. <https://developer.mozilla.org/en-US/docs/Web/CSS/::first-line>.
4. <https://developer.mozilla.org/ru/docs/Web/CSS/::first-line>.

Псевдоклассы для элементов

Помимо псевдоэлементов в CSS существуют псевдоклассы. Они всегда записываются с одним двоеточием после селектора. Псевдоклассы определяют различные состояния элементов, например, их вид при наведении курсора мыши или при клике на элементе. Сегодня мы рассмотрим псевдоклассы, которые отвечают за форматирование определенных элементов в структуре документа.

1 Lorem ipsum dolor sit amet, consectetur adipiscing elit. Illum, ipsum, laudantium. Ipsa possimus consequuntur quasi consequatur veritatis, qui, eligendi mollitia sed non distinctio repellendus alias iste aut, id nam facilis quaerat dolore! Nobis, neque placeat recusandae excepturi magnam. Sit sapiente blanditiis beatae saepe debitis omnis?

2 Facere tempora tenetur minus, inventore, unde illum eveniet nostrum labore fuga magnam iure magni veritatis exercitationem non cum, beatae sed odit veniam deserunt voluptas reiciendis explicabo rerum. Libero quibusdam vero hic, esse sint dicta sed, quidem quas dolore, aut temporibus molestias, officia nobis cumque molestiae.

3 Aspernatur, provident velit atque sint asperiores, quibusdam reiciendis soluta alias nemo eveniet modi cupiditate tempore neque aliquam deserunt deleniti hic sit excepturi. Eius, facilis! Alias id eius dolore minus. Amet rem unde, esse sed harum fugit totam incidunt assumenda quo! Odit libero, eum perspiciatis ipsam.

4 Cumque illum, harum blanditiis consequatur, illo laboriosam sed, non impedit voluptatem excepturi esse. Et eius provident accusantium aperiam tenetur, ad ipsam? Ut, animi ratione rerum cum aliquid corporis porro quibusdam, voluptates cumque? Aspernatur nihil aliquam hic. Veritatis, quis, ut! Quisquam cum maiores explicabo distinctio ducimus.

Иллюстрация 18

Например, псевдокласс `:first-child` отвечает за форматирование первого элемента среди подобных. Например, следующий код приведет к тому, что только первый абзац будет выделен другим цветом и иметь другой шрифт (ил. 18):

```
p:first-child {
  font-family: 'Segoe Script', cursive;
  color: #2c5dcc;
  font-weight: bold;
  border-bottom: 3px double #2c5dcc;
}
```

В противовес **:first-child** псевдокласс **:last-child** отвечает за форматирование последнего элемента среди подобных:

```
p:last-child {
  padding: 15px;
  background-color: #c8fde1;
  border-left: 3px double #32c71e;
}
```

7 Consequuntur officia inventore corporis a aut ipsa molestias, quia ipsum beatae reprehenderit officis est praesentium numquam repellat quo modi minus nisi provident fuga asperiores, sequi labore! Consectetur optio at quaeat est ex molestiae doloreque, quae illum, repellat ratione voluptate sapiente reprehenderit asperiores ipsam, dignissimos ab.

8 Harum qui quibusdam corrupti ratione, enim, veritatis doloreque dicta eos officis sunt sunt quod omnis quas laborum tenetur quaeat soluta obcaecati ullam possimus, animi temporibus sapiente, doloribus. Voluptate eveniet sapiente sunt itaque quia, quam magni, a! A aliquam alias fugiat iure voluptate error ipsam ab!

9 Possimus velit doloribus quasi unde culpa vel pariat voluptatibus minus saepe perferendis nam iste, dignissimos vero, modi optio maxime deleniti reiciendis, amet quidem! Nobis aliquam rem nulla odio ad enim repellendus. Debitis magnam aliquid praesentium, rerum temporibus minima a alias cum quae expedita tempore harum!

10 Quod, accusamus! Dicta molestias, ducimus quibusdam, corrupti sequi incidunt repellendus dolore, qui suscipit nesciunt totam illo accusantium. Id commodi laborum impedit nemo ab obcaecati itaque. Et vitae, consequatur aperiam impedit illo ea quam, beatae quos eos dicta cumque a culpa sequi. Sed, perferendis id expedita.

Иллюстрация 19

Чтобы отформатировать какой-либо определенный элемент, необходимо использовать псевдокласс **:nth-child(n)**.

Вместо **n** нужно поставить какую-либо цифру. Например, для 5-го абзаца код будет выглядеть так:

```
p:nth-child(5) {
    font-style: italic;
    color: #9b9b9b;
}
```

2 Facere tempora tenetur minus, inventore, unde illum eveniet nostrum labore fuga magnam iure magni veritatis exercitationem non cum, beatae sed odit veniam deserunt voluptas reiciendis explicabo rerum. Libero quibusdam vero hic, esse sint dicta sed, quidem quas dolore, aut temporibus molestias, officia nobis cumque molestiae.

3 Aspernatur, provident velit atque sint asperiores, quibusdam reiciendis soluta alias nemo eveniet modi cupiditate tempore neque aliquam deserunt deleniti hic sit excepturi. Eius, facilis! Alias id eius dolorem minus. Amet rem unde, esse sed harum fugit totam incidunt assumenda quo! Odit libero, eum perspiciatis ipsam.

4 Cumque illum, harum blanditiis consequatur, illo laboriosam sed, non impedit voluptatem excepturi esse. Et eius provident accusantium aperiam tenetur, ad ipsam? Ut, animi ratione rerum cum aliquid corporis porro quibusdam, voluptates cumque? Aspernatur nihil aliquam hic. Veritatis, quis, ut! Quisquam cum maiores explicabo distinctio ducimus.

5 Dolore minima optio rem nam facilis ipsam harum nisi atque voluptate laborum assumenda ipsa saepe minus magni eius laudantium, libero temporibus vel eligendi quos? Earum aut explicabo, et voluptates facere dolore nemo qui at saepe esse quaerat expedita placeat quidem veniam fugit iste assumenda neque.

6 In sint dolores ipsam, fugit provident unde itaque nesciunt aliquam illum explicabo ipsum? Quas, labore delectus illo. Vero asperiores tempora deleniti illo nesciunt quos corrupti ipsum ut ullam sequi veniam sed quasi sint repellendus incidunt laborum qui temporibus, iure vitae hic molestiae! Quia, itaque, voluptates.

7 Consequuntur officia inventore corporis a aut ipsa molestias, quia ipsum beatae reprehenderit officis est praesentium numquam repellat quo modi minus nisi provident fuga asperiores, sequi labore! Consectetur optio at quaerat est ex molestiae doloremque, quae illum, repellat ratione voluptate sapiente reprehenderit asperiores ipsam, dignissimos ab.

Иллюстрация 20

Если необходимо задать форматирование для каждого 3-го абзаца, стоит использовать псевдокласс **:nth-child(3n)**. Вместо цифры 3 может быть любая цифра или выражение, например, **4n-1**, **3n+1** и т.д.

```
p:nth-child(3n) {
    color: red;
}
```

1 Lorem ipsum dolor sit amet, consectetur adipiscing elit. Illum, ipsum, laudantium. Ipsa possumus consequuntur quasi consequatur veritatis, qui, eligendi mollitia sed non distinctio repellendus alias iste aut, id nam facilis quaeerat dolorum! Nobis, neque placeat recusandae excepturi magnam. Sit sapiente blanditiis beatae saepe debitis omnis?

2 Facere tempora tenetur minus. inventore, unde illum eveni et nostrum labore fuga magnam iure magna veritatis exercitationem non cum. beatae sed odit veniam deserunt voluptas reiendis explicabo rerum. Libero quibusdam vero hic, esse sint dicta sed, quidem quas dolore, aut temporibus molestias, officia nobis cumque molestiae.

3 Aspernatur provident velit atque sint asperiores, quibusdam reiendis soluta alias nemo eveniet modi cupiditate tempore neque aliquam deserunt deleniti hic sit excepturi. Eius, facilis! Alias id eius dolore minus. Amet rem unde, esse sed harum fugit totam incidunt assumenda quo! Odit libero, eum perspiciatis ipsam.

4 Cumque illum, harum blanditiis consequatur. Illo laboriosam sed, non impedit voluptatem excepturi esse. Et eius provident accusantium aperiam tenetur, ad ipsam? Ut, animi ratione rerum cum aliquid corporis porro quibusdam, voluptates cumque? Aspernatur nihil aliquam hic. Veritatis, quis, ut! Quisquam cum maiores explicabo distinctio duimus.

5 Dolor minima optio rem nam facilis ipsam harum nisi atque voluptate laborum assumenda ipsa saepe minus magni eius laudantium, libero temporibus vel eligendi quis? Earum aut explicabo, et voluptates facere dolore nemo qui at saepe esse quaeerat expedita placeat quidem veniam fugit iste assumenda neque.

6 In sint dolores ipsam, fugit provident unde itaque nesciunt aliquam illum explicabo ipsum? Quas, labore delectus illo. Vero asperiores tempora deleniti illo nesciunt quos corrupti ipsum ut ullam sequi veniam sed quasi sint repellendus incidunt laborum qui temporibus, iure vitae hic molestiae! Quia, itaque, voluptates.

7 Consequuntur officia inventore corporis a aut ipsa molestias, qui a ipsum beatae reprehenderit officis est praesentium numquam repellat quo modi minus nisi provident fuga asperiores, sequi labore! Consectetur optio at quaeerat est ex molestiae doloreque, quae illum, repellat ratione voluptate sapiente reprehenderit asperiores ipsam, dignissimos ab.

8 Harum qui quibusdam corrupti ratione, enim, veritatis doloreque dicta eos officis sint sunt quod omnis quas laborum tenetur quaeerat soluta obcaecati ullam possumus, animi temporibus sapiente, doloribus. Voluptate eveni et sapiente sunt itaque quia, quam magni, a! A aliquam alias fugiat iure voluptate error ipsam ab!

9 Possumus velit doloribus quasi unde culpa vel pariatum voluptatibus minus saepe perferendis nam iste, dignissimos vero, modi optio maxime deleniti reiendis, amet quidem! Nobis aliquam rem nulla odio enim repellendus. Debitis magnam aliquid praesentium, rerum temporibus minima a alias cum quae expedita tempore harum!

Иллюстрация 21

Все примеры вы найдете в файле *pseudoclasses-1.html*.

Достаточно часто бывает нужно отформатировать четные или нечетные элементы. Для этой цели можно использовать, например, псевдокласс `:nth-child(2n)` или `:nth-child(2n-1)`. Кроме того, существуют специальные ключевые слова для четных элементов (`even`) и нечетных (`odd`). В примерах из файла *pseudoclasses-2.html* псевдоклассы заданы для элементов с классом *box*:

```
.box:nth-child(odd) {background-color: #f99;}
```

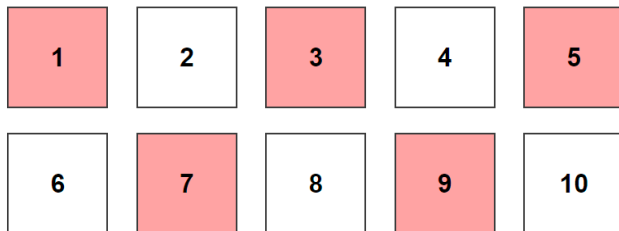


Иллюстрация 22

```
.box:nth-child(even) {background-color: #f8f48c;}
```

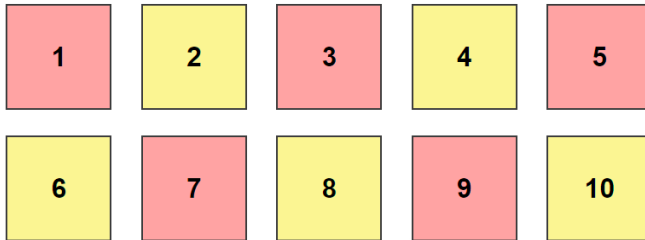


Иллюстрация 23

При написании css-кода Brackets будет подсказывать вам при выборе псевдоклассов. Обращайте, пожалуйста, внимание на такие подсказки — это ускоряет написание кода и предотвращает ошибки (ил. 24, 25).

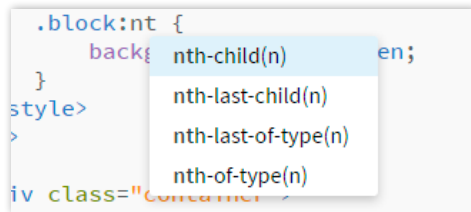


Иллюстрация 24



Иллюстрация 25

Псевдокласс :nth-last-child(n)

Псевдокласс **:nth-last-child(n)** предназначен для форматирования элементов, отсчет которых ведется не с начала их появления в коде, т.е. с первого элемента, а наоборот — с конца, т.е. с последнего элемента.

В скобках вместо **n** можно записывать число, выражение (**2n**, **3n-1**, **4n+1** и т.п.), а также ключевые слова **odd** (нечетные числа) или **even** (четные числа):

```
селектор:nth-last-child( число | выражение | odd | even)
{
    свойства;
}
```

Ссылки по теме:

1. https://www.w3schools.com/cssref/sel_nth-last-child.asp.
2. <https://css-tricks.com/almanac/selectors/n/nth-last-child/>.
3. <http://theory.phphtml.net/css/nth-last-child.html>.
4. <http://htmlbook.ru/css/nth-last-child>.
5. <http://css.yoksel.ru/nth-child/>.

Псевдокласс :only-child

Псевдокласс **:only-child** предназначен для форматирования элементов, которые являются единственными дочерними для своего родительского тега. Т.е. он будет применен в случае, если в родительском элементе для указанного селектора отсутствуют другие элементы такого же типа.

В файле *only-child.html* в папке *examples* вы найдете примеры, иллюстрирующие применение этого псевдокласса.

Первые 2 абзаца содержат внутри теги **<mark>**, предназначенные для выделения текста. По умолчанию он в браузере выделяется желтым фоновым цветом. Если такой тег встречается один раз, мы изменяем его внешний вид с помощью псевдокласса:

```
mark:only-child {
    background-color: #45abf5;
    color: #fff;
    padding: 3px;
    font-family: serif;
    font-style: italic;
}
```

Текст в таком элементе теперь имеет курсивное начертание, белый цвет, голубой фон и гарнитуру с засечками.

Lorem ipsum dolor sit amet, **consectetur adipiscing** elit. Consectetur justo eveniet praesentium, sequi impedit vel, quisquam placeat. Porro, quod natus autem, nam eaque dignissimos dicta unde suscipit necessitatibus incidunt debitis.

Obcaecati minima quas dignissimos ipsa explicabo quam animi aut recusandae beatae, **praesentium mollitia** error accusamus excepturi molestias harum qui, consequuntur **perspiciatis quasi**. Dolor minus cupiditate quisquam expedita, neque perspiciatis **id**.

Иллюстрация 26

В двух последних абзацах файла *only-child.html* вы найдете элементы `<span class="elem">`, для которых цвет текста изменен на красный и выводится он прописными буквами:

```
.elem {
  color: red;
  text-transform: uppercase;
}
```

Но только в абзаце, где расположен один такой элемент, он выделен еще и жирным начертанием:

```
.elem:only-child {
  font-weight: bold;
}
```

Rem reiciendis ea inventore dolore voluptates non adipisci voluptatibus, **TENETUR AB ERROR** delectus ullam magnam necessitatibus unde molestiae soluta. Assumenda minima veritatis nesciunt saepe commodi blanditiis magni. Vero, doloreque, explicabo.

Qui quia magni molestiae **NECESSITATIBUS**, esse possimus iste natus, nisi in repellat ipsam, aliquam facere alias aliquid ea quo soluta **MOLLITIA DICTA** ut nobis nihil nam ratione quasi unde. Laudantium!

Иллюстрация 27

В какой-то степени псевдокласс **:only-child** похож на псевдоклассы **:first-child** или **:last-child**, но они применяются всегда, сколько бы похожих элементов не было, а правила, записанные для **:only-child** сработают только тогда, когда элемент, для которого использован этот псевдокласс, встречается только один раз в своем элементе-контейнере. Т.е. он одновременно будет и первым (**:first-child**), и последним (**:last-child**) «ребенком» для родительского элемента.

Полезные ссылки:

1. <https://developer.mozilla.org/en-US/docs/Web/CSS/:only-child>.
2. https://www.w3schools.com/cssref/sel_only-child.asp.
3. <https://developer.mozilla.org/ru/docs/Web/CSS/:only-child>.
4. https://puzzleweb.ru/css/sel_only-child.php.

Псевдоклассы типа :nth-of-type

Псевдокласс **:nth-of-type(n)** используется для форматирования элементов одного типа внутри одного родительского элемента. Этот псевдокласс может быть записан с цифрой, выражением или ключевыми словами *odd* или *even* в скобках:

```
селектор:nth-of-type( число | выражение | odd | even)
{
    свойства;
}
```

Например, можно записать

```
.block:nth-of-type(3) { свойства; }
p:nth-of-type(3n-1) {свойства; }
.elem:nth-of-type(odd) {свойства; }
```

Кроме того, вы можете использовать псевдоклассы **:first-of-type** или **:last-of-type** для первого и последнего элемента одного типа внутри родительского контейнера.

Также существует псевдокласс **:nth-last-of-type(n)**, который позволяет задавать форматирование для однотипных элементов, отсчитывая их снизу разметки, т.е. от последнего элемента внутри одного контейнера.

Возможно, вы зададитесь вопросом — зачем нужны псевдоклассы, которые делают практически одно и то же?

Почему нельзя ограничиться только лишь псевдоклассами типа `:nth-child`?

Дело в том, что псевдоклассы типа `:nth-child(n)` не всегда ведут себя так, как вы этого ожидаете, особенно при изменении html-разметки. Неверное их поведение с вашей точки зрения, но не с точки зрения браузера, заключается в том, что при добавлении в разметку новых элементов к тем, для которых вы задали некие свойства, используя `:nth-child(n)` и его разновидности (`:first-child`, `:last-child`, `:only-child`), вы заметите, что ситуация с форматированием элементов изменилась.

Давайте рассмотрим практический пример, чтобы стало понятнее, о чем идет речь.

Предположим, что нам нужно создать 4 блока с одинаковым форматированием, но с разным цветом фона. В файле *blocks-nth-child.html* из папки *examples* вы найдете форматирование 4-х блоков следующего вида:

```
.block {  
    display: inline-block;  
    width: 21%;  
    height: 150px;  
    line-height: 170%;  
    font-size: 2rem;  
    font-weight: bold;  
    text-align: center;  
    margin: 2%;  
    border: 2px solid;  
}
```

Цвет фона будем задавать с помощью псевдоклассов `:first-child`, `:nth-child(n)`, `:last-child`:

```
.block:first-child {
    background-color: green;
}
.block:nth-child(2) {
    background-color: #32c132;
}
.block:nth-child(3) {
    background-color: #aff062;
}
.block:last-child {
    background-color: #dee863;
}
```

Внешний вид файла показан на скриншоте (ил. 28).



Иллюстрация 28

Все элементы с классом *block* на данный момент расположены в родительском элементе с классом *container*:

```
<div class="container">
    <div class="block"><h2>Block 1</h2></div>
    <div class="block"><h2>Block 2</h2></div>
    <div class="block"><h2>Block 3</h2></div>
    <div class="block"><h2>Block 4</h2></div>
</div>
```

Допустим, нам необходимо добавить к ним заголовок (довольно распространенная ситуация), который также будет размещен внутри элемента с классом *container*. Код изменится таким образом:

```

<div class="container">
  <h1>Our blocks</h1>
  <div class="block"><h2>Block 1</h2></div>
  <div class="block"><h2>Block 2</h2></div>
  <div class="block"><h2>Block 3</h2></div>
  <div class="block"><h2>Block 4</h2></div>
</div>

```

Внешний вид блоков в браузере изменится (ил. 28).

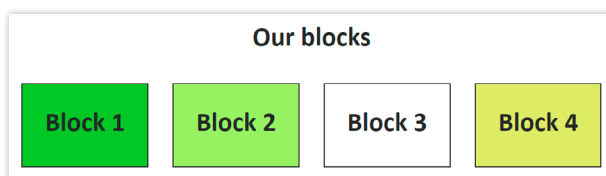


Иллюстрация 29

Что же произошло? Нагляднее будет разобраться с помощью рисунка (ил. 30).

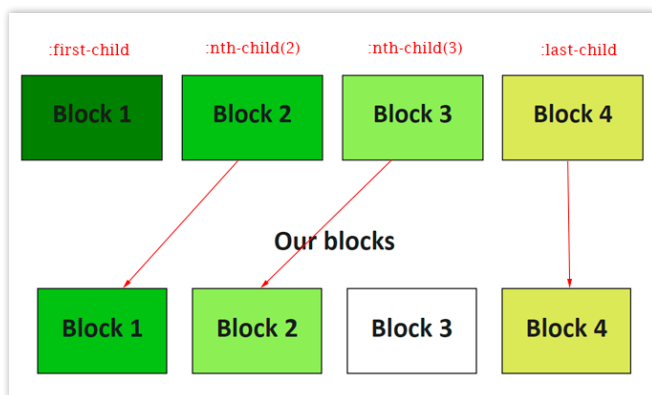


Иллюстрация 30

Стрелками показаны «переехавшие» css-правила. Можно заметить, что произошло смещение на один блок

влево. Т.е. элемент, который подчинялся правилам для `:first-child` теперь подчиняется правилам для элемента с псевдоклассом `:nth-child(2)`, а элемент, для которого свойства описывал псевдокласс `:nth-child(2)` теперь имеет свойства элемента с псевдоклассом `:nth-child(3)`. Третий блок вообще потерял цвет фона. Неизменным осталось только форматирование для `:last-child`.

Почему поменялись «дети» в родительском элементе с классом *container*? Ответ прост — именно потому, что «детей» стало больше. При добавлении заголовка в элементе с классом *container* увеличилось количество дочерних, или вложенных, элементов (*child elements* на английском), причем заголовок стал в иерархии дочерних элементов первым, т.е. псевдокласс `:first-child` теперь должен относиться именно к нему. Однако `css`-правила записаны для `.block:first-child`, поэтому к заголовку не применяются. Если бы правила были написаны для `.container>:first-child`, тогда бы мы увидели темно-зеленый фон у заголовка.

По той же причине **Block 1** стал управляться правилами для `.block:nth-child(2)`, а **Block 2** — правилами для `.block:nth-child(3)`. Поскольку после последнего блока мы не добавляли никаких новых элементов, то для него по-прежнему актуально форматирование в виде правил для `.block:last-child`. Давайте изменим и это, добавив в элемент с классом *container* элемент `<footer>` в нижней части:

```
<div class="container">
  <h1>Our blocks</h1>
  <div class="block"><h2>Block 1</h2></div>
  <div class="block"><h2>Block 2</h2></div>
```

```
<div class="block"><h2>Block 3</h2></div>
<div class="block"><h2>Block 4</h2></div>
<footer>&copy; nth-child blocks</footer>
</div>
```

Теперь цвет фона «потерял» и 4-й блок, т.к. последним ребенком (**:last-child**) для элемента с классом *container* стал элемент **<footer>**, а форматирование назначено для *.block:last-child*.

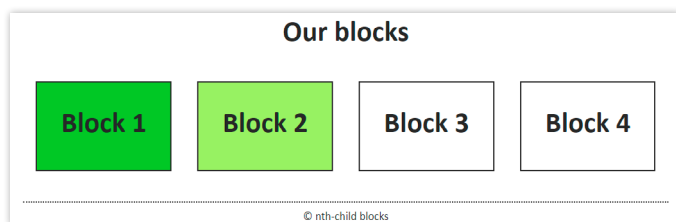


Иллюстрация 31

Ситуацию поможет изменить замена форматирования, заданного псевдоклассами **:nth-child** и подобных ему, на **:nth-of-type**. Аналогичный пример, но с другими псевдоклассами вы найдете в файле *blocks-nth-of-type.html*. Код стилей с псевдоклассами таков:

```
.block:first-of-type {
    background-color: green;
}
.block:nth-of-type(2) {
    background-color: #32c132;
}
.block:nth-of-type(3) {
    background-color: #aff062;
}
```

```
.block:last-of-type {  
    background-color: #dee863;  
}
```

Кроме того, и заголовков, и элемент **footer** (*подвал*) остались в html-разметке. Тем не менее, все блоки в этом файле имеют разный цвет фона, причем в таком виде, в котором он был задан в первом файле ДО добавления заголовка и подвала.

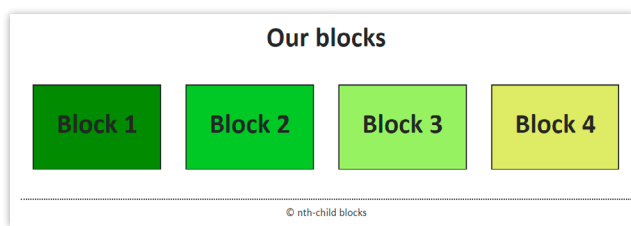


Иллюстрация 32

Из этого примера можно сделать вывод о том, что форматирование дочерних элементов с помощью псевдоклассов типа **:nth-child** подходит в том случае, если родительский элемент содержит элементы одного типа — с одинаковыми тегами или классами, например. Если же в разметке родительского контейнера присутствуют разнотипные элементы, нужно использовать псевдоклассы типа **:nth-of-type**.

Использование псевдоклассов при верстке блока страницы

Давайте рассмотрим еще один пример, в котором нам нужно будет отформатировать 4 подобных блока с учетом использования разных цветов для текста, фона и рамок. Кроме того, блоки будут расположены со смещением относительно друг друга по вертикали.

Внешний вид блоков показан на скриншоте (ил. 33), текст для блоков вы найдете в файле *pseudoclasses-example-text.txt*, а готовый пример — в файле *pseudoclasses-example.html*.



Иллюстрация 33

Для начала поменяйте расширение файла с txt на html. Это можно сделать в Brackets, нажав клавишу **F2**. Только после этого выделим весь текст (**CTRL + A**) и нажмем сочетание клавиш **CTRL + SHIFT + A**.

В открывшемся внизу поле с заголовком *Enter Abbri-
viation* введем восклицательный знак (!) и получим базовую структуру html-документа.

Меняем текст в теге **<title>** и продолжаем использовать сочетание клавиш **CTRL + SHIFT + A** для других текстовых элементов.

```

1 <!DOCTYPE html>
2 <html lang="en">
3 <head>
4   <meta charset="UTF-8">
5   <title>Document</title>
6 </head>
7 <body>
8   Adobe Photoshop
9   Experience with different multi-layer psd-
10  layouts.
11  Output images from the psd-layout.
12  Image optimization for faster uploading.
13  Skill 1
14
15  HTML5/CSS3
16  Layout of valid html-pages
17  Understanding the principles of responsive

```

Enter Abbri- viation |

Строка 5, Столбец 20 — Выделено 8 столбцов — 38 строк

ВСТ UTF-8 HTML Пробелы: 4

Иллюстрация 34

Поскольку текст каждого из 4-х блоков содержит 2 заголовка (в начале и конце) и маркированный список из 3-4-х пунктов, воспользуемся оборачиванием для однотипных элементов в каждом блоке, выделив соответствующий текст с зажатой клавишей **CTRL**.

```

7 <body>
8   Adobe Photoshop
9   Experience with different multi-layer psd-layouts.
10  Output images from the psd-layout.
11  Image optimization for faster uploading.
12  Skill 1
13
14  HTML5/CSS3
15  Layout of valid html-pages
16  Understanding the principles of responsive design layout,
17  adaptive page layout
18  Bootstrap based html-page layout
19  Good experience in creating multi-screen landing pages.
20  Skill 2
21
22  JavaScript/jQuery
23  Understanding the DOM model in html document
24  Experience using jQuery plugins: sliders, lightboxes, and
25  others.
26  Customization of scripts, setting of plugin parameters
27  Initial knowledge of Vue.js
28  Skill 3
29
30  PHP/MySQL
31  Creating plug-ins for Wordpress.
32  Using PHP/MySQL to create themes for Wordpress.
33  Using MVC Templates.
34  Initial knowledge of E-commerce Systems
35  Skill 4

```

Иллюстрация 35

После нажатия клавиш **CTRL + SHIFT + A** вводим в поле с *Enter Abbreviation* **h2** и получаем 4 заголовка второго уровня:

```

5   <title>Document</title>
6   </head>
7   <body>
8     <h2>Adobe Photoshop</h2>
9     Experience with different multi-layer psd-layouts.
10    Output images from the psd-layout.
11    Image optimization for faster uploading.
12    Skill 1
13
14    <h2>HTML5/CSS3</h2>
15    Layout of valid html-pages
16    Understanding the principles of responsive design layout,
17    adaptive page layout
18    Bootstrap based html-page layout
19    Good experience in creating multi-screen landing pages.
20    Skill 2

```

Иллюстрация 36. (Начало)

```

20
21 <h2>JavaScript/jQuery</h2>
22 Understanding the DOM model in html document
23 Experience using jQuery plugins: sliders, lightboxes, and
24 others.
25 Customization of scripts, setting of plugin parameters
26 Initial knowledge of Vue.js
27 Skill 3
28 <h2>PHP/MySQL</h2>
29 Creating plug-ins for Wordpress.
30 Using PHP/MySQL to create themes for Wordpress.
31 Using MVC Templates.
32 Initial knowledge of E-commerce Systems
33
Enter Abbreviation [h2]

```

Иллюстрация 36. (Продолжение)

Аналогичным образом выделяем текст «Skill» с цифрой и оборачиваем его в теги заголовка 3-го уровня **h3**.

```

6 </head>
7 <body>
8 <h2>Adobe Photoshop</h2>
9 Experience with different multi-layer psd-layouts.
10 Output images from the psd-layout.
11 Image optimization for faster uploading.
12 <h3>Skill 1</h3>
13
14 <h2>HTML5/CSS3</h2>
15 Layout of valid html-pages
16 Understanding the principles of responsive design layout,
17 adaptive page layout
18 Bootstrap based html-page layout
19 Good experience in creating multi-screen landing pages.
20 <h3>Skill 2</h3>
21
22 <h2>JavaScript/jQuery</h2>
23 Understanding the DOM model in html document
24 Experience using jQuery plugins: sliders, lightboxes, and
25 others.
26 Customization of scripts, setting of plugin parameters
27 Initial knowledge of Vue.js
28 <h3>Skill 3</h3>
29
30 <h2>PHP/MySQL</h2>
31 Creating plug-ins for Wordpress.
32 Using PHP/MySQL to create themes for Wordpress.
33 Using MVC Templates.
34 Initial knowledge of E-commerce Systems
35 <h3>Skill 4</h3>
36
Enter Abbreviation [h3]

```

Иллюстрация 37

Можно еще улучшить нашу аббревиатуру, добавив в нее html-спецсимвол **»**, который обозначает

символ правых двойных кавычек и используется в наших блоках (ил. 38).



Иллюстрация 38

В этом случае аббревиатура для оборачивания тегов меняется на `h3>{» ;}`.

Обратите внимание на наличие пробела внутри фигурных скобок — это тот текст, который будет добавлен перед словом «Skill».

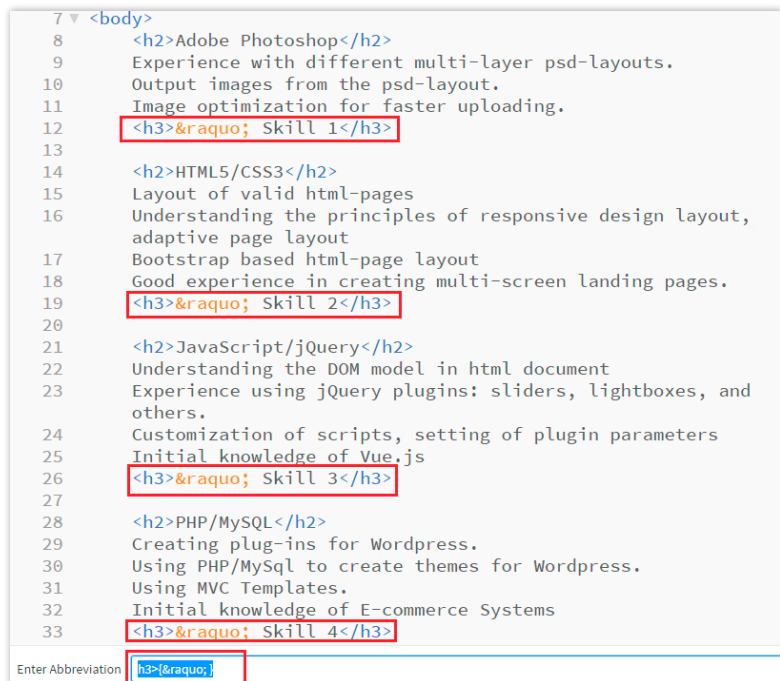


Иллюстрация 39

При использовании спецсимвола мы получим только одну кавычку. Вторую будем добавлять с помощью псевдоэлемента `::before`, чтобы получить представление об использовании еще одного способа.

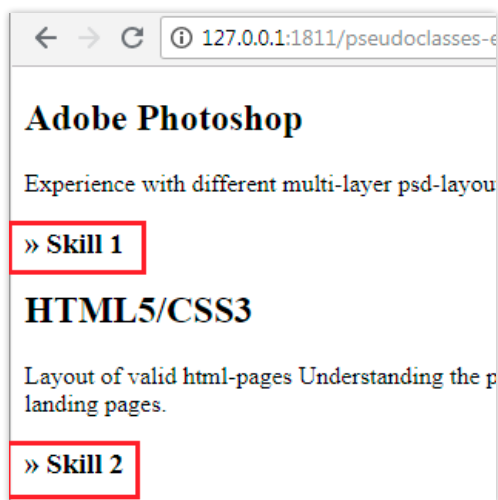


Иллюстрация 40

Еще один заголовок, на этот раз первого уровня, находится в самом низу. Сначала добавляем в конце спецсимвол `»`, а затем оборачиваем его в теги `<h1>`.

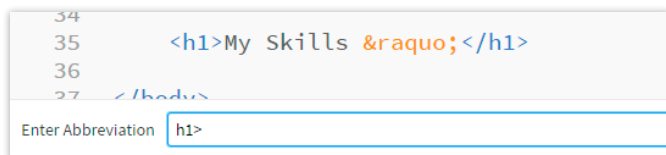


Иллюстрация 41

Теперь необходимо обернуть в теги списков все оставшиеся строки текста.

Выделяем их с помощью зажатой клавиши **CTRL**.

```

8      <h2>Adobe Photoshop</h2>
9      Experience with different multi-layer psd-layouts.
10     Output images from the psd-layout.
11     Image optimization for faster uploading.
12     <h3>&raquo; Skill 1</h3>
13
14     <h2>HTML5/CSS3</h2>
15     Layout of valid html-pages
16     Understanding the principles of responsive design layout,
17     adaptive page layout
18     Bootstrap based html-page layout
19     Good experience in creating multi-screen landing pages.
20     <h3>&raquo; Skill 2</h3>
21
22     <h2>JavaScript/jQuery</h2>
23     Understanding the DOM model in html document
24     Experience using jQuery plugins: sliders, lightboxes, and
25     others.
26     Customization of scripts, setting of plugin parameters
27     Initial knowledge of Vue.js
28     <h3>&raquo; Skill 3</h3>
29
30     <h2>PHP/MySQL</h2>
31     Creating plug-ins for Wordpress.
32     Using PHP/MySQL to create themes for Wordpress.
33     Using MVC Templates.
34     Initial knowledge of E-commerce Systems
35     <h3>&raquo; Skill 4</h3>

```

Иллюстрация 42

В поле *Enter Abbreviation* вводим аббревиатуру с тегами списков и знаком * в конце, который позволит нам добавить теги `<li>` ко всем строкам, которые сформированы в текстовом документе с помощью перевода строки.

```
ul>li*
```

```

8      <h2>Adobe Photoshop</h2>
9      <ul>
10         <li>Experience with different multi-layer psd-layouts.</li>
11         <li>Output images from the psd-layout.</li>
12         <li>Image optimization for faster uploading.</li>
13     </ul>
14     <h3>&raquo; Skill 1</h3>

```

Иллюстрация 43. (Начало)

```

15      <h2>HTML5/CSS3</h2>
16      <ul>
17        <li>Layout of valid html-pages</li>
18        <li>Understanding the principles of responsive design
19          layout, adaptive page layout</li>
20        <li>Bootstrap based html-page layout</li>
21        <li>Good experience in creating multi-screen landing pages.
22        </li>
23      </ul>
24      <h3>&raquo; Skill 2</h3>
25      <h2>JavaScript/jQuery</h2>
26      <ul>
27        <li>Understanding the DOM model in html document</li>
28        <li>Experience using jQuery plugins: sliders, lightboxes,
29          and others.</li>
30        <li>Customization of scripts, setting of plugin
31          parameters</li>
32        <li>Initial knowledge of Vue.js</li>

```

Enter Abbreviation

Иллюстрация 43. (Продолжение)

Теперь оборачиваем часть уже сформированной разметки от тега `<h2>` до закрывающего тега `</h3>` включительно в `div` с классом `box`:

```
.box
```

Процедуру повторяем 4 раза.

```

7 <body>
8   <div class="box">
9     <h2>Adobe Photoshop</h2>
10    <ul>
11      <li>Experience with different multi-layer psd-layouts.
12      </li>
13      <li>Output images from the psd-layout.</li>
14      <li>Image optimization for faster uploading.</li>
15    </ul>
16    <h3>&raquo; Skill 1</h3>
17  </div>
18  <div class="box">
19    <h2>HTML5/CSS3</h2>
20    <ul>
21      <li>Layout of valid html-pages</li>

```

Иллюстрация 44. (Начало)

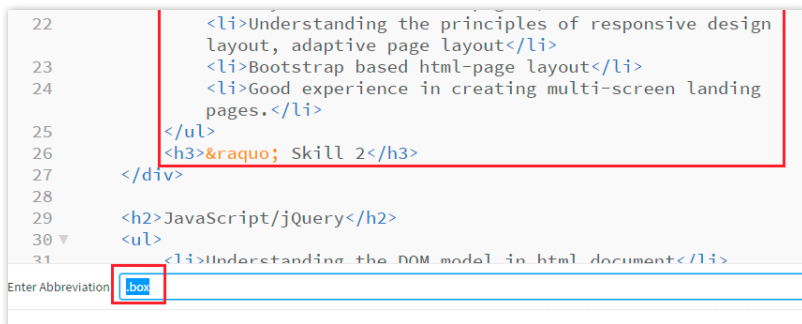


Иллюстрация 44. (Продолжение)

Последняя наша работа с обращиванием текста и тегов в аббревиатуру заключается в том, что нужно выделить все элементы между тегами `<body>` `</body>` и заключить их в `div` с классом *container*:

```
.container // Emmet-аббревиатура
```



Иллюстрация 45

Внешний вид страницы после добавления всех тегов таков (ил. 46):

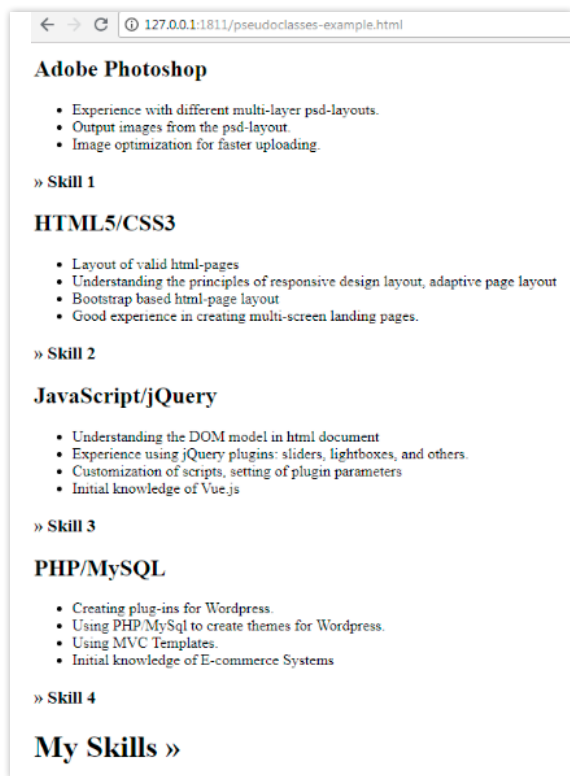


Иллюстрация 46

Теперь наступает очередь css-форматирования. В тегах `<style>` в блоке `<head>...</head>` разместим следующий код:

```
*, *::before, *::after {
    box-sizing: border-box;
}
body { font-family: sans-serif; }
.container {
    width: 90%;
    max-width: 1200px;
    min-width: 768px;
```

```

margin: 20px auto;
}
h1, h2, h3 {
  font-family: 'Bookman Old Style', serif;
  letter-spacing: 1px;
}
.box {
  width: 48%;
  border: 2px solid;
  padding: 4%;
}

```



Иллюстрация 47

На данный момент мы сообщили браузеру, что размеры всех элементов, а также псевдоэлементов `::before`

и `::after` у нас будут включать внутренние отступы и рамки, указали семейство шрифта для `body` и минимальную и максимальную ширину для `div.container`, а также изменили шрифт и отступ между буквами (`letter-spacing`) для заголовков. Для `div.box` указали размеры, стиль и толщину рамки, а также внутренние отступы. Получили в браузере 4 блока, плотно примыкающие друг к другу.

Теперь нам необходимо распределить их по 2 слева и 2 справа и назначить цвет фона и текста. Выполнять это будем с помощью псевдоклассов типа `:nth-child`. Все нечетные блоки (`.box:nth-child(odd)`) разместим слева с помощью CSS-свойства `float: left`, а все четные блоки (`.box:nth-child(even)`) — справа с помощью `float: right`:

```
.box:nth-child(odd) {  
    float: left;  
}  
.box:nth-child(even) {  
    float: right;  
}
```

Получим такой вид файла в браузере (ил. 48).

Добавим внешние отступы к нашим нечетным и четным блокам с классом `box`:

```
.box:nth-child(odd) {  
    float: left;  
    margin-bottom: 50px;  
}  
.box:nth-child(even) {  
    margin-top: 50px;  
    float: right;  
}
```

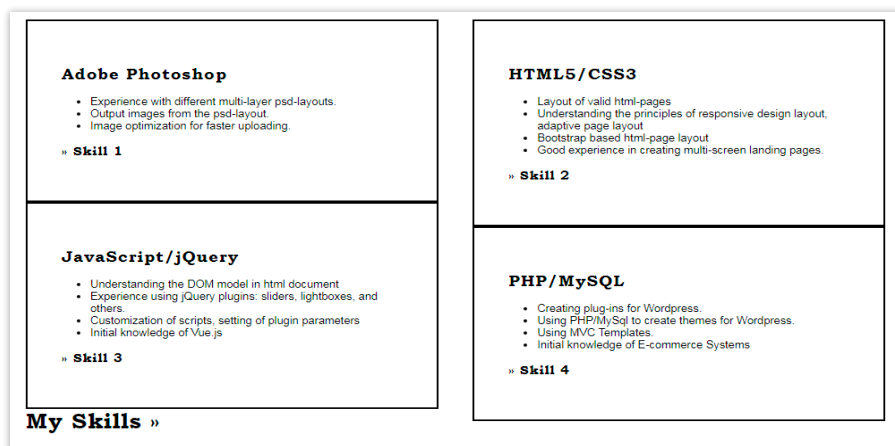


Иллюстрация 48

Результат представлен на скриншоте (ил. 49).

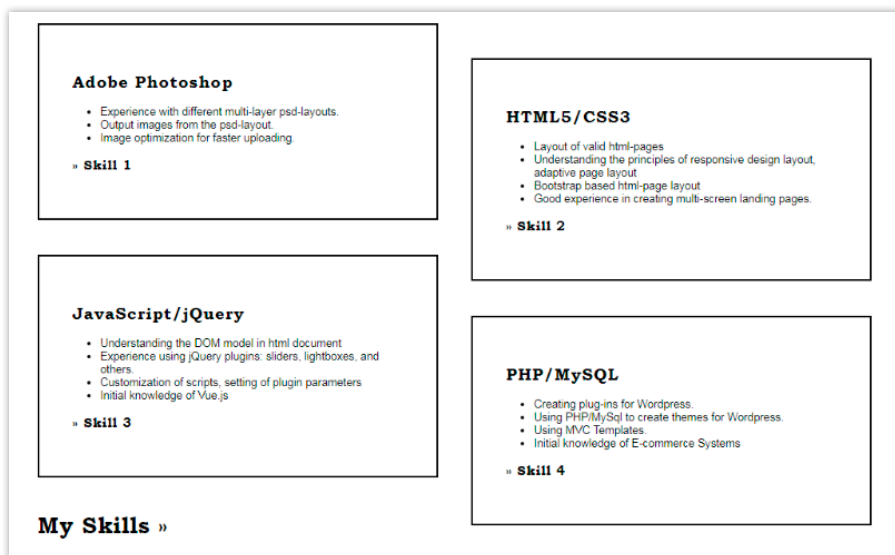


Иллюстрация 49

Заметьте, что мы использовали 2 селектора для того, чтобы разместить 4 блока.

Теперь необходимо добавить цветовое оформление. Его тоже будем задавать с помощью псевдоклассов:

```
.box:first-child {
    color: #79c6ff;
    background-color: darkblue;
}
.box:nth-child(2), .box:nth-child(3) {
    background-color: #ffea91;
    color: #773b08;
}
.box:last-child {
    color: #450080;
    background-color: #b864ff;
}
```

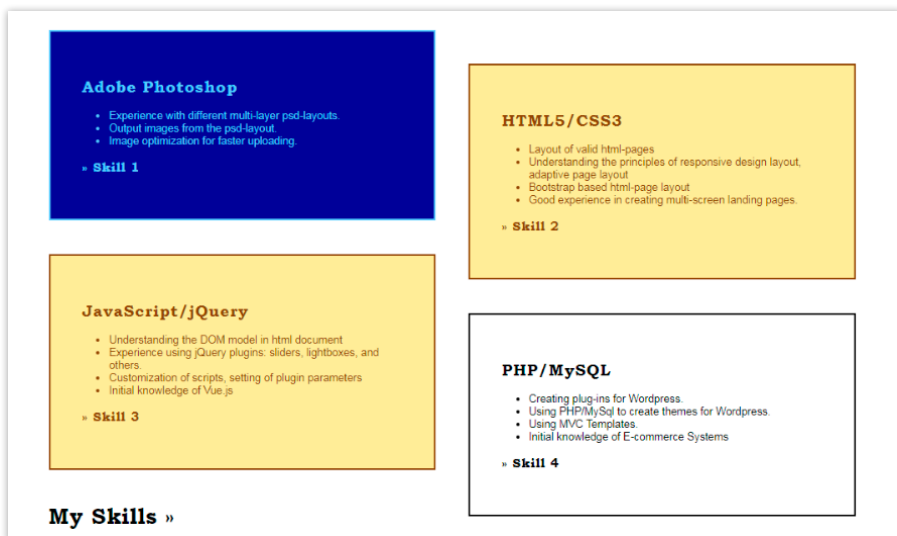


Иллюстрация 50

Обратите внимание, что при указании одинаковых правил для разных блоков, через запятую перечисляются не как `nth-child(2,3)`, а как `.box:nth-child(2)`, `.box:nth-child(3)`.

Внешний вид блоков в браузере поменялся (ил. 50).

Три блока из четырех действительно поменяли цвет, и только последний остался без форматирования. В чем же дело?

На скриншоте (ил. 51) представлены все цвета, которые были использованы для указания цвета текста и цвета фона для каждого из блоков. Обратите внимание, что мы не задаем цвет рамок, т.к. по умолчанию они совпадают с цветом текста для блока.

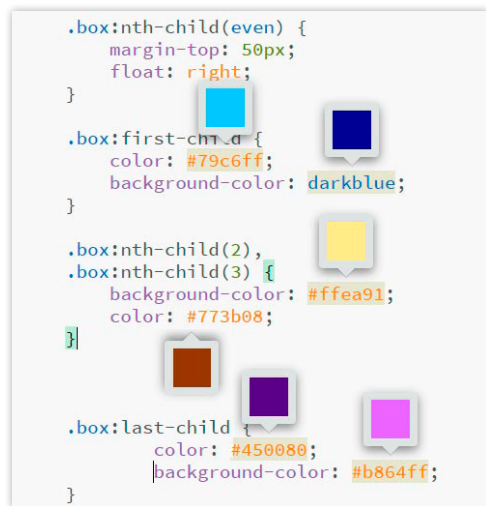


Иллюстрация 51

Из этого скриншота видно, что цвета первого, второго и третьего блока совпадают, а у последнего блока должен быть сиреневый цвет фона и фиолетовый цвет текста. Значит, дело не в неверно назначенных цветах. Проблема заключается в неверно заданном селекторе. Поскольку в нашей разметке после последнего `div` с классом `box` идет заголовок первого уровня, то `div.box` НЕ является

последним ребенком для родительского элемента `div` с классом `container`, т.к. им является заголовок:

```
100 ▶      <div class="box"> ... </div>
110
111      <h1>My Skills &raquo;</h1>
112      </div>
```

Иллюстрация 52

Поэтому меняем селектор с `.box:last-child` на `.box:last-of-type`, т.к. нам нужно отформатировать последний элемент типа `.box`:

```
.box:last-of-type {
  color: #450080;
  background-color: #b864ff;
}
```

Теперь страница выглядит как нужно (ил. 53).

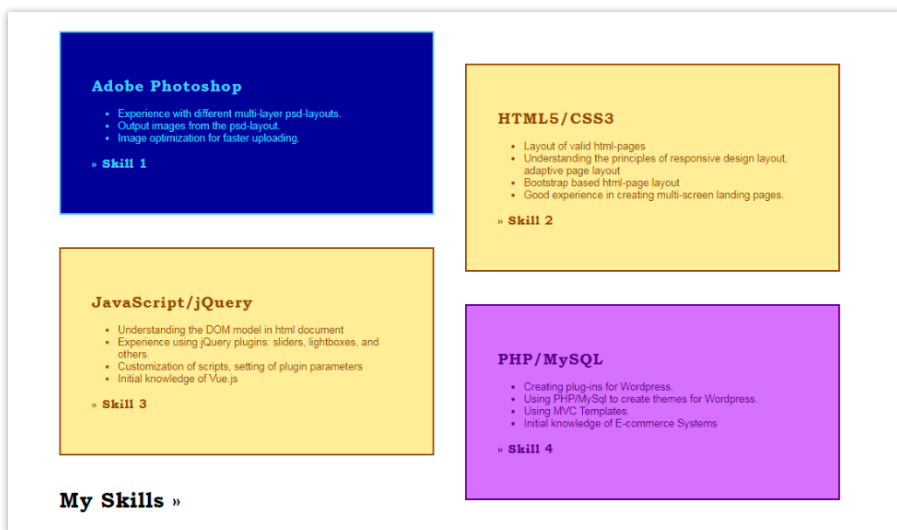


Иллюстрация 53

Добавим немного оформления к нашим блокам с классом *box*.

```
.box {  
    width: 48%;  
    border: 2px solid;  
    padding: 4%;  
    outline: 2px solid;  
    outline-offset: 4px;  
}
```

Свойство **outline** подобно свойству **border**, а именно оно задает внешнюю границу для элемента за пределами **border**, но при этом оно не влияет на ширину элемента. В плане записи это свойство является универсальным, или составным, т.е. состоит из 3-х свойств:

```
outline: outline-color || outline-style ||  
        outline-width | inherit
```

Все составляющие свойства аналогичны таким же для составного свойства **border**, т.е. для **border-color**, **border-style** и **border-width**, которые мы рассматривали в прошлом уроке. В отличие от **border**, **outline** можно задать сразу для всех сторон, для каждой по отдельности — нельзя. Это свойство в силу того, что оно не добавляет дополнительных пикселей в размеры элемента (ширина, высота), используют для выделения элементов форм и ссылок при переходе фокуса на них.

Мы также используем свойство **outline-offset**, которое устанавливает расстояние между рамкой от свойства **outline** и границ элемента. Минусом этого свойства

является то, что его не в полной мере поддерживают браузеры. В данном случае мы используем его для того, чтобы познакомиться с этим свойством.

IE	Edge	Firefox	Chrome	Safari	Opera	QQ Safari	Opera Mini	Android	BlackBerry	Opera Mobile	Uptime Android	Firefox Android	IE Mobile	Uuiter Android	Samsung Internet	QQ	Beau
		49	56	5	41	4.1											
		50	57	5.1	42	4.3											
		51	58	6	43	5.1											
		52	59	6.1	44	6.1		2.1									
		53	60	7	45	7.1		2.2									
		54	61	7.1	46	8		2.3									
		55	62	8	47	8.4		3									
		56	63	9	48	9.2		4									
6	12	57	64	9.1	49	9.3		4.1									
7	13	58	65	10	50	10.2		4.3									
8	14	59	66	10.1	51	10.3		4.4		12					4		
9	15	60	67	11	52	11.2		4.4		12.1					5		
10	16	61	68	11.1	53	11.4		4.4	7	12.1			10		6.2		
11	17	62	69	12			all	67	10	46	67	60	11	11.8	7.2	1.2	7.12
	18	63	70	TP													
		63	70	TP													
		71															

Иллюстрация 54

Итак, мы получили следующее отображение элементов с классом *box* (ил. 55).

Adobe Photoshop

- Experience with different multi-layer psd-layouts.
- Output images from the psd-layout.
- Image optimization for faster uploading.

» Skill 1

JavaScript/jQuery

- Understanding the DOM model in html document
- Experience using jQuery plugins: sliders, lightboxes, and others.
- Customization of scripts, setting of plugin parameters
- Initial knowledge of Vue.js

» Skill 3

HTML5/CSS3

- Layout of valid html-pages
- Understanding the principles of responsive design layout, adaptive page layout
- Bootstrap-based html-page layout
- Good experience in creating multi-screen landing pages.

» Skill 2

PHP/MySQL

- Creating plug-ins for Wordpress.
- Using PHP/MySQL to create themes for Wordpress.
- Using MVC Templates.
- Initial knowledge of E-commerce Systems

» Skill 4

My Skills »

Иллюстрация 55

Теперь добавим тень:

```
.box {
  ...;
  box-shadow: 6px 6px 16px rgba(0, 0, 0, 0.4);
}
```



Иллюстрация 56

Такой вид тени очень слабо виден на фоне 2-х рамок, сформированных свойствами **border**, **outline** и **outline-offset**. Поэтому сделаем 2 тени: одну белую, без смещения, которая займет расстояние в 6px (2px **outline-width** + 4px **outline-offset**), а также увеличим смещение темной тени:

```
.box {
  width: 48%;
  border: 2px solid;
  padding: 4%;
```

```
outline: 2px solid;
outline-offset: 4px;
box-shadow: 0 0 0 6px #fff,
           12px 12px 16px rgba(0, 0, 0, 0.4);
}
```



Иллюстрация 57

Следующей задачей является оформление заголовков. Сейчас мы можем видеть, что отступы до заголовка второго и третьего уровня находятся слишком далеко от рамки.

```
.box h2, .box h3 {
    font-size: 1.4em;
    text-align: right;
}

.box h2 { margin-top: 0; }
.box h3 { margin-bottom: -2%; }
```

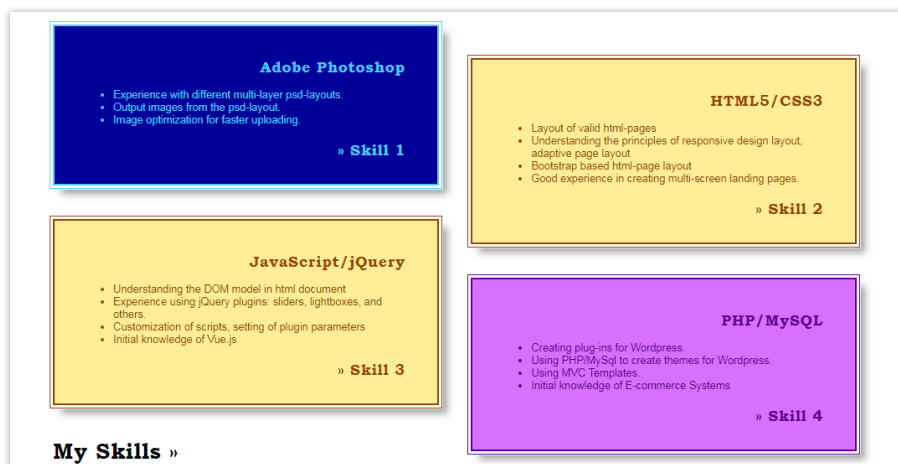


Иллюстрация 58

Добавим еще символ двойных правых кавычек. Для чего конвертируем описание спецсимвола в 16-ричный код. Можно найти соответствие символу `»` на css-tricks.com, нажав **CTRL + F**, или воспользоваться конвертером на xiper.net, в который необходимо вставить 10-ричный код спецсимвола вида `»` без знаков `&#`.

<code>&raquo;</code>	»	<code>&#187;</code>	angle quotation mark, right	<code>%BB</code>	<pre>p:before { content:"\00bb"; }</pre>	<code>alert("\273");</code>
--------------------------	---	-------------------------	-----------------------------	------------------	--	-----------------------------

Иллюстрация 59

Для того, чтобы облегчить тебе перевод (а заодно и подбор подходящего спецсимвола) мы сделали специальный калькулятор (в инпуты вводим только числовой код, без знаков `&#`):

В десятичной системе:

В шестнадцатеричной системе:

Вид спецсимвола:

»

Иллюстрация 60

В итоге получим такой код:

```
h3:before, h1:after {
    content: '\bb ';
}
```

В каждом заголовке **h3** и **h1** либо до текста (**h3:before**), либо после (**h1:after**) добавился второй символ двойной правой кавычки.

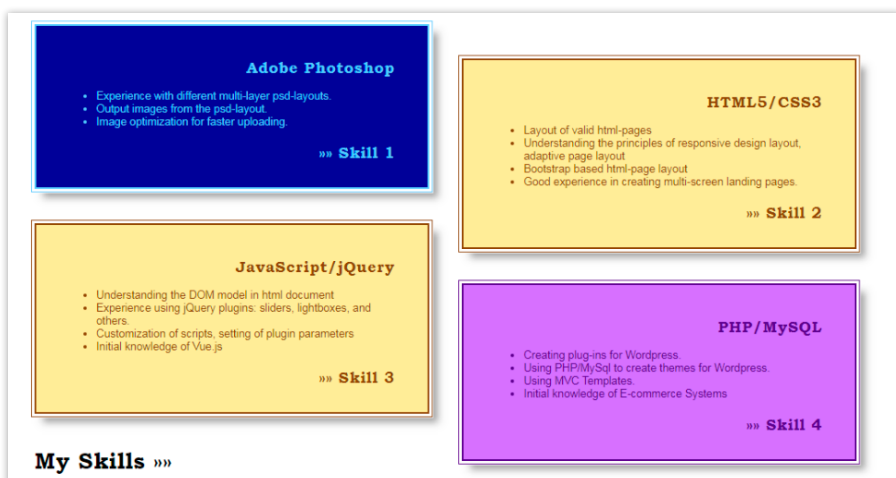


Иллюстрация 61

Примечание. Этот способ можно использовать там, где у вас нет по каким-либо причинам доступа к html-разметке, но есть доступ к css-правилам, например, при настройке шаблона CMS (системы управления контентом типа Wordpress, Joomla или Drupal).

Последнее, что нам осталось — это очистка обтекания, о которой мы говорили в прошлом уроке. Если мы

сейчас зададим серый фоновый цвет для заголовка первого уровня, то получим интересную картину.

```
h1{background-color: #ccc;}
```

Оказывается, заголовок распространился под всю область наших блоков. Это очень хорошо видно на скриншоте (ил. 62).

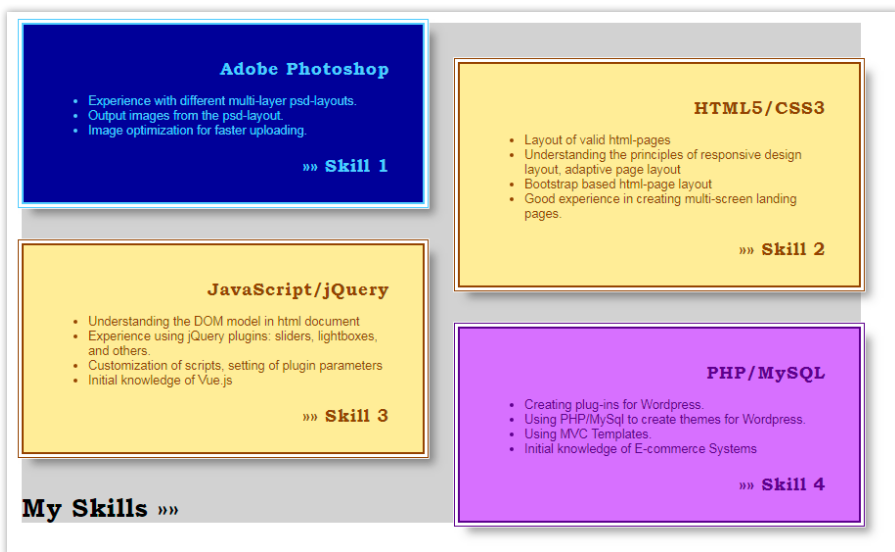


Иллюстрация 62

Поскольку элемент `h1` находится в самом низу нашей разметки, имеет смысл назначить отмену обтекания с помощью свойства `clear: both` именно для этого элемента:

```
h1 {
  background-color: #ccc;
  clear: both;
}
```

Adobe Photoshop

- Experience with different multi-layer psd-layouts.
- Output images from the psd-layout.
- Image optimization for faster uploading.

»» Skill 1

JavaScript/jQuery

- Understanding the DOM model in html document
- Experience using jQuery plugins: sliders, lightboxes, and others.
- Customization of scripts, setting of plugin parameters
- Initial knowledge of Vue.js

»» Skill 3

HTML5/CSS3

- Layout of valid html-pages
- Understanding the principles of responsive design layout, adaptive page layout
- Bootstrap based html-page layout
- Good experience in creating multi-screen landing pages.

»» Skill 2

PHP/MySQL

- Creating plug-ins for Wordpress.
- Using PHP/MySQL to create themes for Wordpress.
- Using MVC Templates.
- Initial knowledge of E-commerce Systems

»» Skill 4

My Skills »»

Иллюстрация 63

Adobe Photoshop

- Experience with different multi-layer psd-layouts.
- Output images from the psd-layout.
- Image optimization for faster uploading.

»» Skill 1

JavaScript/jQuery

- Understanding the DOM model in html document
- Experience using jQuery plugins: sliders, lightboxes, and others.
- Customization of scripts, setting of plugin parameters
- Initial knowledge of Vue.js

»» Skill 3

HTML5/CSS3

- Layout of valid html-pages
- Understanding the principles of responsive design layout, adaptive page layout
- Bootstrap based html-page layout
- Good experience in creating multi-screen landing pages.

»» Skill 2

PHP/MySQL

- Creating plug-ins for Wordpress.
- Using PHP/MySQL to create themes for Wordpress.
- Using MVC Templates.
- Initial knowledge of E-commerce Systems

»» Skill 4

My Skills »»

Иллюстрация 64

Теперь заголовок принял свои обычные размеры по размеру текста. Уберем выделение фоновым цветом (ил. 64), оставив одно правило:

```
h1 { clear: both; }
```

Итоговый вариант файла *pseudoclasses-example.html*.

Полезные ссылки:

1. https://www.w3schools.com/cssref/sel_nth-of-type.asp.
2. <https://developer.mozilla.org/en-US/docs/Web/CSS/:nth-of-type>.
3. <https://habr.com/post/119139/>.
4. <https://webref.ru/css/outline>.
5. <https://css-tricks.com/snippets/html/glyphs/>.
6. <http://xiper.net/collect/html-and-css-tricks/content/insert-symbols>.
7. <http://www.web.cofp.ru/vse-o-sajtaxh/sozdanie-sajta/verstka/110-special-chars>.

Создание колонок с помощью строчно-блочных элементов

Вернемся к строчно-блочным элементам и рассмотрим, каким образом мы можем их использовать при форматировании различных элементов html-страницы.

Еще одним преимуществом строчно-блочных элементов по сравнению со строчными является возможность создавать колонки при назначении им свойства `width`. В силу своей «линейности» эти элементы выстраиваются в колонки друг рядом с другом без использования обтекания с помощью свойства `float: left` или `right` и собственных им недостатков (обнуления высоты родительского элемента, обтекания ниже идущими элементами).

Здесь, правда, тоже не обошлось без сюрпризов. Особенностью строчно-блочных элементов является то, что браузер рассчитывает их ширину с учетом одного пробельного отступа после закрывающего тега. Величина этого отступа зависит от размера и семейства используемого шрифта и обычно составляет 3-5px при стандартном размере шрифта в 16px. Если вы хотите использовать строчно-блочные элементы для создания колонок, то вам придется обязательно учитывать этот отступ при верстке колонок.

В этом разделе мы рассмотрим 2 способа верстки части страницы в виде колонок и целой страницы в виде сетки с помощью элементов с `display: inline-block`, а также способы «избавления» от пробельного отступа.

Первый пример вы найдете в файле *column-inline-block.html* в папке *examples* данного урока. В нем мы будем создавать 4 колонки.

Для начала используем аббревиатуру *Emmet* для формирования html-разметки блока с колонками:

```
.wrap>h1{The Best Offer}+.product*4>h2.  
product-title{Product $}+.product-descr>p*2>lorem12^^.  
price[data-price]{Price}+a.btn[#]>{Add to cart}
```

Это достаточно сложная аббревиатура создаст элемент *div* с классом *wrap*, внутри которого разместятся заголовок первого уровня и 4 *div*-а с классом *product*. Каждый из «продуктовых» *div*-ов имеет заголовок второго уровня с классом *product-title* и текстом «Product» с порядковым номером от 1 до 4-х, который создает плагин *Emmet* из знака \$ и *4. После заголовка следует *div* с классом *product-descr* с двумя абзацами шаблонного текста из 12 слов. После описания продукта (*product-descr*) в родительском элементе *.product* следует *div* с классом *price*, у которого после названия класса в квадратных скобках указан атрибут *data-price*. Мы будем его использовать для формирования цены с помощью псевдоэлемента *::after*. В самом конце аббревиатуры разместилась ссылка с классом *btn* и текстом «Addtocart».

Все это длинное описание вы можете быстро просмотреть, если нажмете клавишу **Tab** в конце аббревиатуры:

```
<div class="wrap">  
  <h1>The Best Offer</h1>  
  <div class="product">  
    <h2 class="product-title">Product 1</h2>
```

```

    <div class="product-descr">
      <p>Lorem ipsum dolor sit ...</p>
      <p>Perferendis cum accusantiumducimus,
        eamagni...
      </p>
    </div>

    <div class="price" data-price="22.99">
      Price
    </div>

    <a href="#" class="btn">Add to cart</a>
  </div>

  <div class="product">
    <h2 class="product-title">Product 2</h2>

    <div class="product-descr">
      <p>Lorem ipsum dolor sit amet...</p>
      <p>Voluptatum corporis ...</p>
    </div>

    <div class="price" data-price="14.99">
      Price
    </div>

    <a href="#" class="btn">Add to cart</a>
  </div>

  ...
</div>

```

В этой разметке уже добавлены цифры в атрибуте *data-price*. Вы можете их расставить самостоятельно.

На данный момент разметка выглядит очень просто в браузере:

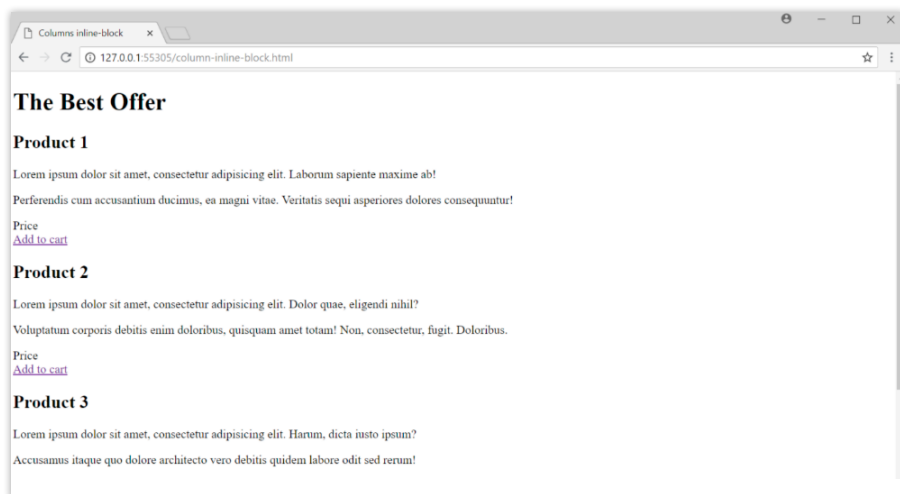


Иллюстрация 65

Давайте зададим css-форматирование:

```
body {
    font-family: Verdana, Geneva, sans-serif;
    background-color: #bcbcbc;
}
.wrap {
    max-width: 1200px;
    width: 90%;
    min-width: 768px;
    margin: 20px auto;
}
h1 {
    text-align: center;
    margin-bottom: 2em;
}
```

Поскольку такие правила форматирования встречались нам достаточно часто, я не буду их комментировать. Изменения на странице выглядят так:

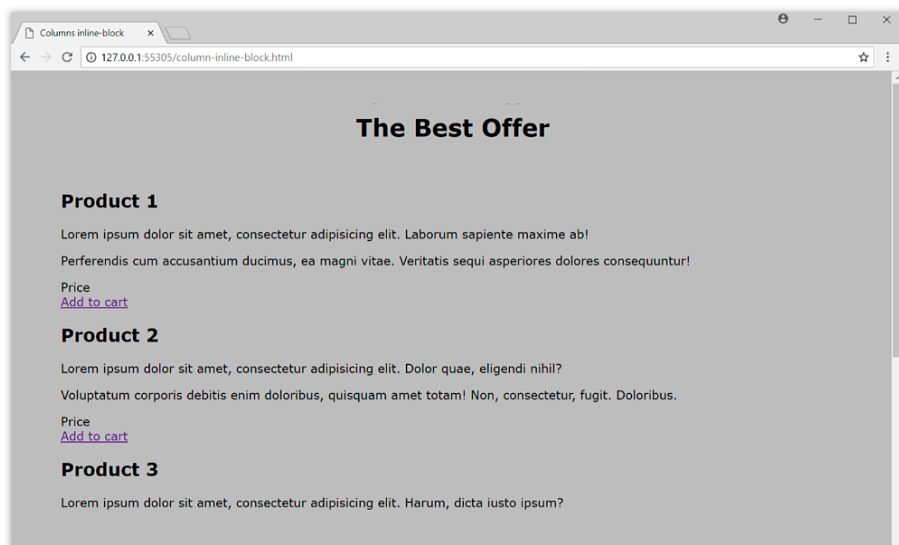


Иллюстрация 66

Ширина `div`-а с классом `wrap` для вложенных в него элементов составляет 100%. Т.е. для каждого из продуктов, которых у нас 4, ширина должна быть 25%, рассчитанных по такой простой формуле: $100\% : 4 = 25\%$.

Используем полученное значение для `css`-правил класса `product`:

```
.product {
  display: inline-block;
  width: 25%;
  background-color: #fff;
}
```

Получим интересную ситуацию, при которой между блоками появилось небольшое расстояние, хотя `margin` мы для них не задавали, но при этом последняя колонка «переехала» на нижнюю строку.

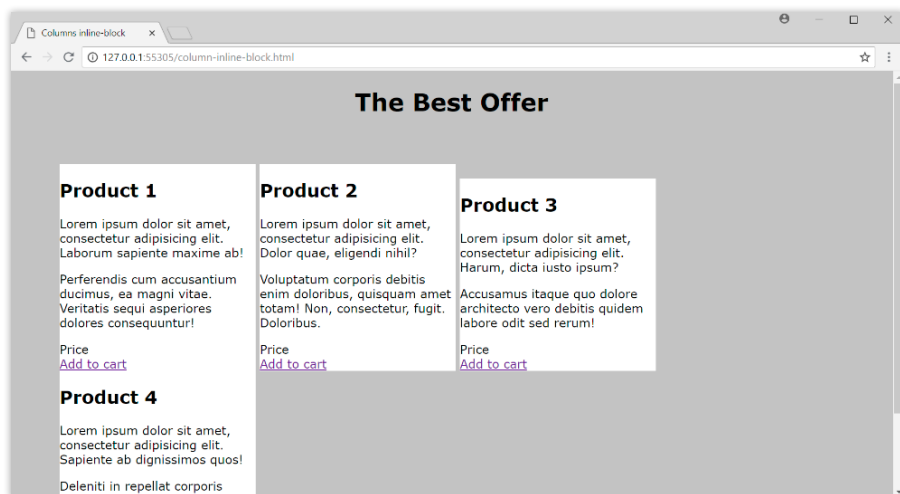


Иллюстрация 67

В чем же дело? Почему так произошло? Вся создавшаяся картина связана с отображением строчно-блочных элементов, а именно с тем, что закрывающий тег (в нашем случае `</div>` для класса *product*) и перевод строки после него, выполненный клавишей **Enter**, в HTML преобразуются в один пробел для любых строчных или строчно-блочных элементов. Именно его мы и наблюдаем между колонками. Его величина составляет примерно 5px или $0.25\text{--}0.3\text{em}$ для выбранного нами шрифта. Однако, и такой малости хватает, чтобы сдвинуть последнюю колонку вниз.

Для того чтобы убрать этот отступ, воспользуемся функцией `calc()`, которая появилась CSS3:

```
.product {
  display: inline-block;
  width: calc(25% - .27em);
  background-color: #fff;
}
```

Ситуация с колонками изменилась — теперь они помещаются в один ряд, причем при любом разрешении экрана.

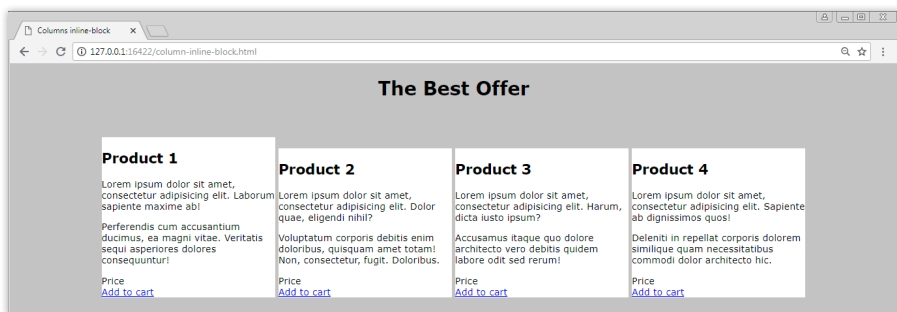


Иллюстрация 68

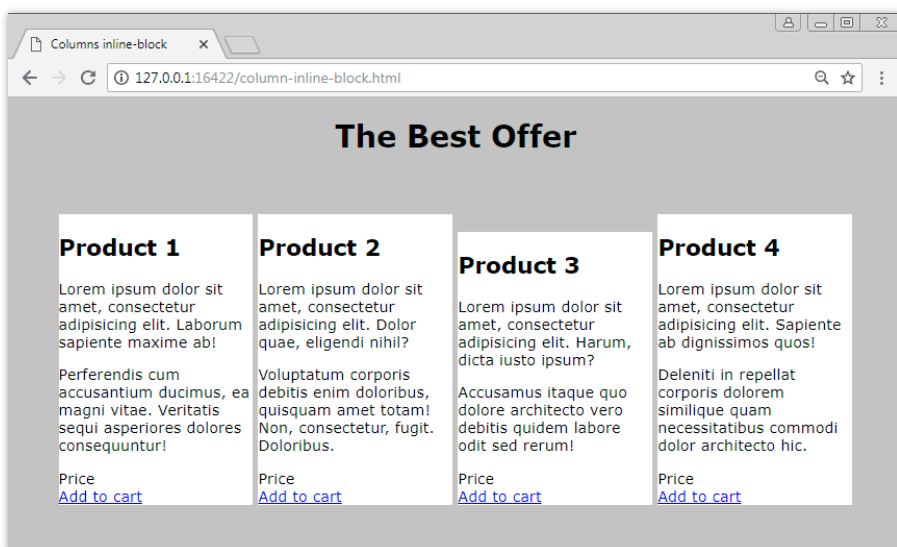


Иллюстрация 69

Теперь желательно выровнять блоки с продуктами по верхней линии. Для этого предназначено свойство **vertical-align** со значением **top**:


```
.product {
  display: inline-block;
  width: calc(25% - .27em);
  background-color: #fff;
  vertical-align: top;
}
```

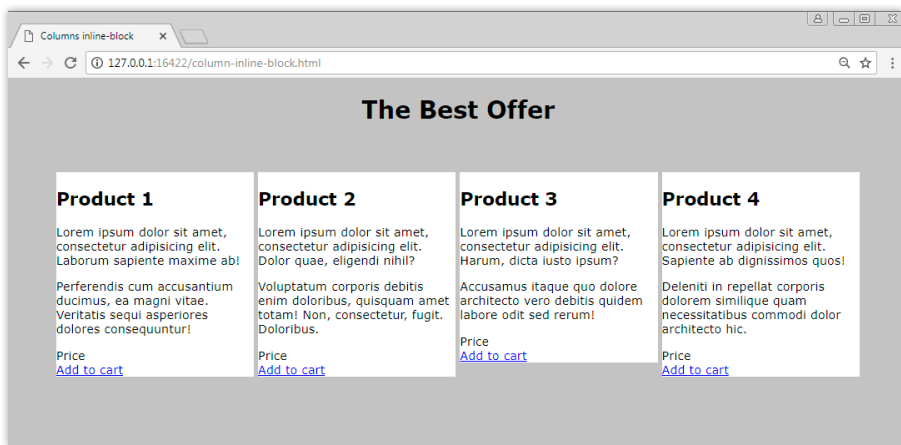


Иллюстрация 70

Для улучшения внешнего вида добавим внутренний отступ (**padding**) к классу *product*.

```
.product {
  display: inline-block;
  width: calc(25% - .27em);
  background-color: #fff;
  vertical-align: top;
  padding: 0 15px;
}
```

После этого опять получим «переезд» последней колонки вниз. Теперь это смещение связано с увеличением

расчетной ширины колонки в браузере за счет того, что реальная ширина складывается из свойства **width** и свойства **padding**, т.е. к величине 25% -.27em прибавились 30px внутренних отступов, и колонки опять перестали вкладываться в 100% ширины родительского элемента с классом *wrap*.

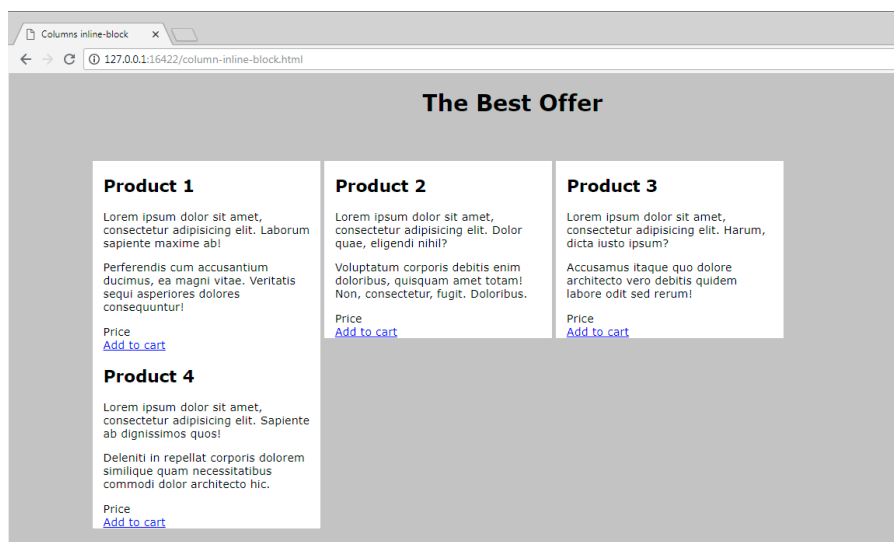


Иллюстрация 71

Эта проблема решается с помощью уменьшения значения свойства **width** или с помощью **box-sizing: border-box**:

```
.product {
  display: inline-block;
  width: calc(25% - 30px - .27em);
  background-color: #fff;
  vertical-align: top;
  padding: 0 15px;
}
```

ИЛИ:

```
.product {
  box-sizing: border-box;
  display: inline-block;
  width: calc(25% - .27em);
  background-color: #fff;
  vertical-align: top;
  padding: 0 15px;
}
```

Второй способ является более универсальным, поэтому в коде оставим его.



Иллюстрация 72

Отступы в один пробельный отступ, пожалуй, слишком малы для разделения наших колонок. Поэтому добавим отступ справа (**margin-right**) в 2% и на эту величину уменьшим ширину (**width**) колонки. Также добавим отступ снизу с помощью **margin-bottom**:

```
.product {
  box-sizing: border-box;
  display: inline-block;
  width: calc(23% - .27em);
  background-color: #fff;
```

```
vertical-align: top;
padding: 0 15px;
margin-right: 2%;
margin-bottom: 20px;
}
```

Сейчас все вроде бы в порядке. Однако если открыть инспектор свойств, то можно заметить, что справа от последней колонки есть лишнее пространство. Это **margin-right** у последней колонки.

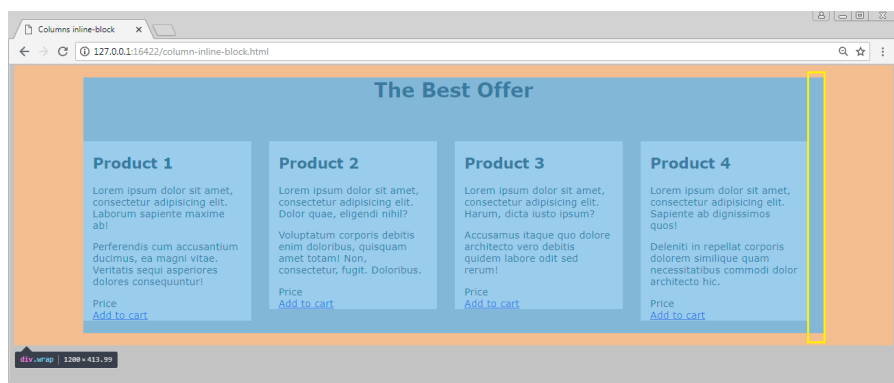


Иллюстрация 73

Для нашего примера это не слишком важно, но если на странице будет несколько разделов с такими же размерами, этот отступ будет бросаться в глаза. Поэтому нам нужно перераспределить пространство отступов между колонками таким образом, чтобы после последней колонки отступа не было.

Для этого определим, что нам нужны 3 отступа по 2%. Соответственно на колонки у нас останется $100\% - 3 \times 2\% = 94\%$. Ширина каждой колонки будет $94\% / 4 = 23.5\%$.

Не стоит забывать об использовании функции `calc()` для свойства `width`:

```
.product {
  box-sizing: border-box;
  display: inline-block;
  width: calc(23.5% - .27em);
  background-color: #fff;
  vertical-align: top;
  padding: 0 15px;
  margin-right: 2%;
  margin-bottom: 20px;
}
```

Кроме того, чтобы в очередной раз у нас не съехала последняя колонка, необходимо отключить для нее `margin-right`, задав для этого свойства значение 0. Сделаем это с помощью псевдокласса `:last-child`:

```
.product:last-child {
  margin-right: 0;
}
```

Теперь все 4 колонки четко вписываются в пределы родительского элемента:



Иллюстрация 74

Далее мы оформляем заголовок и описание продукта:

```
.product-title {
  text-align: center;
  border-bottom: 2px dashed;
  padding-bottom: 10px;
}

.product-descr {
  font-family: Cambria, serif;
}
```

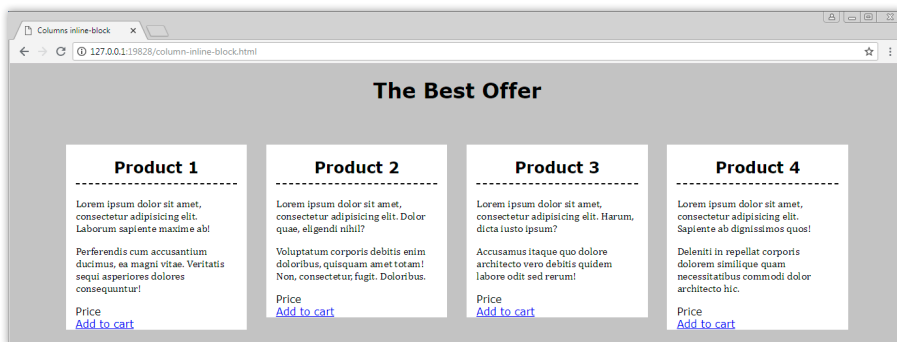


Иллюстрация 75

Для класса *price* используем оформление с помощью псевдокласса *::after*. Напомню, что в html-разметке мы использовали data-атрибут *data-price* с различными значениями для разных блоков с классом *product*:

```
82 <div class="wrap">
83   <h1>The Best Offer</h1>
84   <div class="product">
85     <h2 class="product-title">Product 1</h2>
86     <div class="product-descr">...</div>
87     <div class="price" data-price="22.99">Price</div>
88     <a href="#" class="btn">Add to cart</a>
89   </div>
90   <div class="product">
```

Иллюстрация 76. (Начало)

```

94      <h2 class="product-title">Product 2</h2>
95 ▶    <div class="product-descr">... </div>
100    <div class="price" data-price="14.99">Price</div>
101    <a href="#" class="btn">Add to cart</a>
102  </div>
103 ▼  <div class="product">
104    <h2 class="product-title">Product 3</h2>
105 ▶  <div class="product-descr">... </div>
110    <div class="price" data-price="16.59">Price</div>
111    <a href="#" class="btn">Add to cart</a>
112  </div>

```

Иллюстрация 76. (Продолжение)

Для свойства **content**, которое обязательно нужно указывать в псевдоэлементе **::after** доступна функция **attr()**, внутри скобок которой указывают тот атрибут, значение которого нужно подставить в функцию:

```

.price:after {
  content: attr()
}

```

Иллюстрация 77

Добавим перед функцией знак **\$** в кавычках и получим такие правила:

```

.price:after {
  content: "$attr(data-price);
  color: red;
}

```

Product 1	Product 2	Product 3	Product 4
Lorem ipsum dolor sit amet, consectetur adipiscing elit. Laborum sapiente maxime ab! Perferendis cum accusantium ducimus, ea magni vitae. Veritatis sequi asperiores dolores consequuntur! Price\$22.99 Add to cart	Lorem ipsum dolor sit amet, consectetur adipiscing elit. Dolor quae, eligendi nihil? Voluptatum corporis debitis enim doloribus, quisquam amet totam! Non, consectetur, fugit. Doloribus. Price\$14.99 Add to cart	Lorem ipsum dolor sit amet, consectetur adipiscing elit. Harum, dicta iusto ipsum? Accusamus itaque quo dolore architecto vero debitis quidem labore odit sed rerum! Price\$16.59 Add to cart	Lorem ipsum dolor sit amet, consectetur adipiscing elit. Sapiente ab dignissimos quos! Deleniti in repellat corporis dolorem similique quam necessitatibus commodi dolor architecto hic. Price\$17.99 Add to cart

Иллюстрация 78

За счет использования разных значений в атрибуте `data-price` для разных колонок мы получаем разнообразие величин цены товара.

Примечание. На самом деле мы могли создать какой-либо `span` в `div`-е с классом `price` и записать в нем значение для цены товара, а затем задать для `span` красный цвет текста. Тем не менее, `data`-атрибуты с ценой вполне могут быть в реальном коде, но, скорее всего, в них будет записана цена продукта в различных валютах, например:

```
<div class="price" data-dollar="14.99" data-rub="
959.36" data-uah="419.72" >Price <span class=
"price-value" > $14.99 <span> </div>
```

С помощью JavaScript эти данные можно подставить в `span` с классом `price-value` и показать пользователю при выборе другой валюты на сайте без перезагрузки страницы.

Для `div`-а с классом `price` установим жирное начертание шрифта и добавим отступ снизу, а текст псевдоэлемента `::after` сместим вправо с помощью свойства `float: right`:

```
.price {
  font-weight: bold;
  margin-bottom: 10px;
}
.price::after {
  content: "$"attr(data-price);
  color: red;
  float: right;
}
```

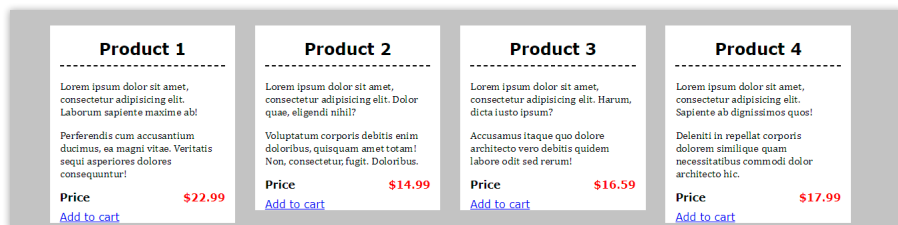



Иллюстрация 79

Последним элементом, требующим форматирования, является ссылка с текстом «Add to cart» и классом *btn*. Для нее запишем следующие css-правила:

```
.btn {
  text-decoration: none;
  background-color: #f00;
  color: #fff;
  display: block;
  width: 100px;
  padding: 7px 20px;
  border-bottom: 2px solid black;
  margin: 15px auto 20px;
  text-align: center;
}
```

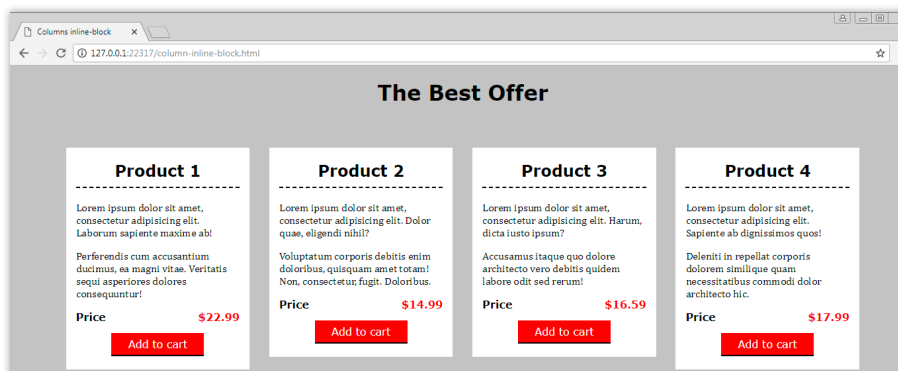


Иллюстрация 80

Последним штрихом в форматировании этих блоков является подгонка высоты колонок с классом *product*. Воспользуемся для этого Инспектором свойств (**F12** или **CTRL + SHIFT + I**). Из скриншота (ил. 81) видно, что для больших экранов минимальная высота блока должна быть примерно 335px.

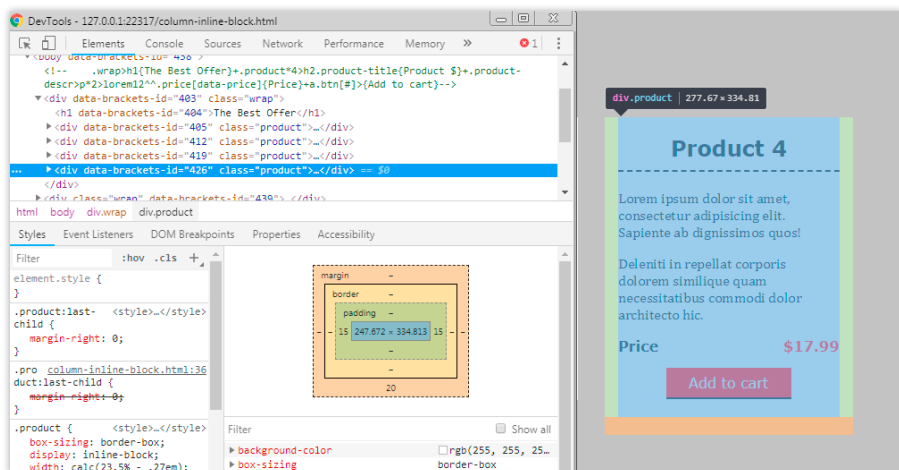


Иллюстрация 81

Используем это значение для свойства **min-height**:

```
.product {
  box-sizing: border-box;
  display: inline-block;
  width: calc(23.5% - .27em);
  min-height: 335px;
  background-color: #fff;
  vertical-align: top;
  padding: 0 15px;
  margin-right: 2%;
  margin-bottom: 20px;
}
```

Результат форматирования представлен на скриншоте (ил. 82).

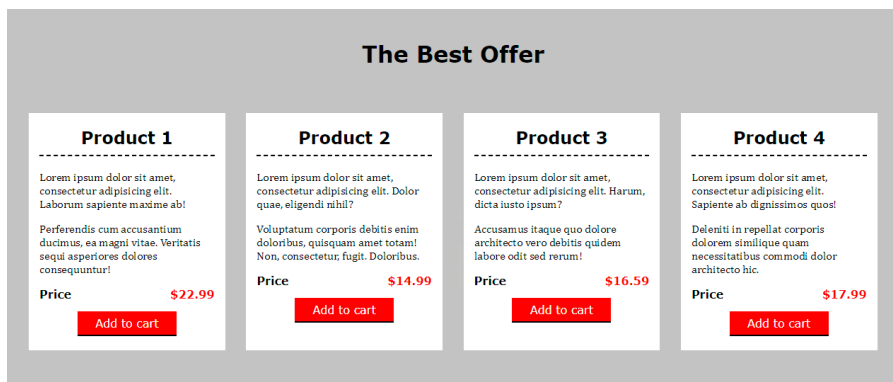


Иллюстрация 82

Полезные ссылки:

1. <https://developer.mozilla.org/en-US/docs/Web/CSS/calc>.
2. <https://css-tricks.com/a-couple-of-use-cases-for-calc/>.
3. <https://developer.mozilla.org/en-US/docs/Web/CSS/vertical-align>.
4. https://www.w3schools.com/cssref/pr_pos_vertical-align.asp.
5. <https://web-standards.ru/articles/vertical-align/>.
6. <http://html-plus.in.ua/osobennosti-blokov-s-display-in-line-block/>.
7. <http://htmlbook.ru/blog/zhemchuzhina-css3-funkciya-calc>.
8. <http://css-live.ru/articles/kogda-byvaet-nuzhen-calc.html>.

Использование строчно-блочных элементов для верстки страницы

Рассмотрим, как можно сверстать простую адаптивную страницу на основе inline-block-элементов. В данном примере адаптивность будет определяться использованием единиц измерений для свойств `width` и `height` в % и `vmax` (`vh`), которые переводятся браузером в px в зависимости от ширины и высоты окна. Код и внешний вид страницы можно посмотреть в файле *display-inline-block-grid.html* в папке *examples* урока.



Иллюстрация 83

Для разметки используем ряд элементов `article` с классами `grid-item` и `item` с цифрой, т.к. предполагается, что на странице будут размещены вводные данные о нескольких статьях с ссылками на основной контент. Мы поместим их в основной элемент `main`, который понадобится для назначения некоторых `CSS`-свойств.

```
<main>
  <article class="grid-item item1">
    <h4 class="place">California</h4>
    <h2 class="post-title">Jumping Around</h2>
    <p>California's last empty jump spots</p>
    <a href="#">Discover more</a>
  </article>

  <article class="grid-item item2">
    <h4 class="place">Italy</h4>
    <h2 class="post-title">Calm Sirentity</h2>
    <p>Italy's secrets meadows and fields</p>
    <a href="#">Check them out</a>
  </article>
  ...
</main>
```

Аббревиатуру Emmet для создания основы кода вы найдете в комментариях в теле `html`-документа *display-inline-block-grid.html* или можете скопировать ниже:

```
main>article.grid-item.item$*6>h4.place+h2.
post-title+p+a[#]{Read more}
```

Что касается использования 2-х классов для одного элемента, то это не только допустимо для любого элемента, но и часто используется в верстке. В нашем случае первый

класс — *grid-item*—необходим для назначения общих правил форматирования статей, а класс *item* с цифрой — для назначения разного фонового цвета.

```
.grid-item {  
    display: inline-block;  
    width: 50%;  
    min-width: 370px;  
    height: 50vmax;  
}  
  
.item1 {background-color: #ff0;}  
.item2 {background-color: #f00;}  
.item3 {background-color: #0f0;}
```

CSS-правила для класса *.grid-item* задают отображение изначально блочного элемента *article* как строчно-блочного с шириной в 50% (т.е. половина от доступного пространства родительского элемента *main*). Для того чтобы контент красиво смотрелся на разных экранах, минимальную ширину (*min-width*) назначим в 370px, а для вертикального деления страницы на 2 блока зададим высоту статьи (*height*) в 50vmax. Подробнее почитать о единицах *vmax* можно в статьях, хотя они рассматривались в материале первого урока:

1. <https://web-design-weekly.com/2014/11/18/viewport-units-vw-vh-vmin-vmax/>
2. <https://webformyself.com/minimum-i-maksimum-ponyatiya-vmin-i-vmax-v-css/>

Однако внешний вид страницы далек от запланированного нами (ил. 84).



Иллюстрация 84

Почему так произошло? Во-первых, элемент `body` по умолчанию имеет внешние отступы в 8px. Во-вторых, элементы `article` со свойством `display: inline-block` имеют 1 пробельный отступ, который не позволяет разместить 2 элемента рядом по горизонтали, как это делается для блочных элементов, имеющих свойство `float` со значением `left` или `right`. Мы видели этот отступ в предыдущем примере.

Для того чтобы убрать отступы в `body` и у заголовков, создадим в папке `examples` папку `css`, разместим в ней файл `reset.css`, созданный Эриком Мейером, а потом добавим на страницу код из него. Для этого в блоке `<head>` после тега `<title>` добавим тег `<link>`:

```
<link rel="stylesheet" href="css/reset.css">
```

Аббревиатура Emmet для создания этого тега — `link`.

В атрибуте `href` этого тега записывается путь к css-файлу, который нужно загрузить для форматирования html-страницы. После добавления ссылки на этот файл в теге `link` все css-правила, описанные в нем, применяются к элементам нашей страницы.

Если внимательно сравнить 2 скриншота (ил. 85, 86), то можно заметить, что после применения правил из файла `reset.css` исчезли не только отступы у `body` и заголовков, но еще заголовки «потеряли» жирность шрифта + все элементы стали иметь одинаковый размер шрифта.



Иллюстрация 85

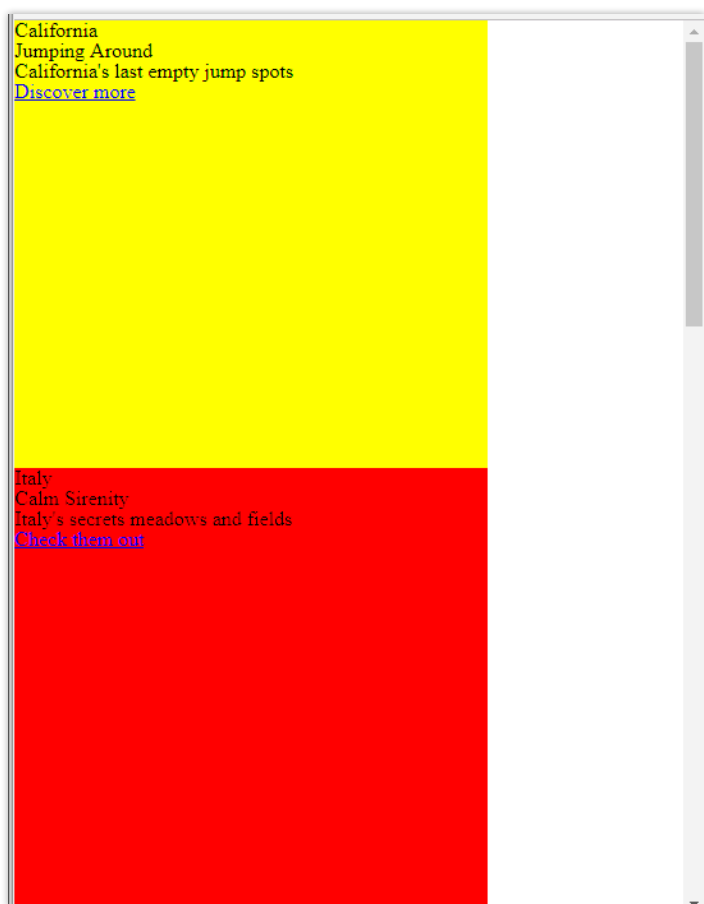


Иллюстрация 86

Тем не менее, цветные блоки все еще располагаются друг под другом. Чтобы этого избежать, добавим для класса *grid-item* отрицательный внешний отступ справа:

```
.grid-item {  
    ...; /* все предыдущие CSS-свойства */  
    margin-right: -.25em;  
}
```

Теперь можно увидеть, как элементы *article* с классом *.grid-item* разместились рядом друг с другом:

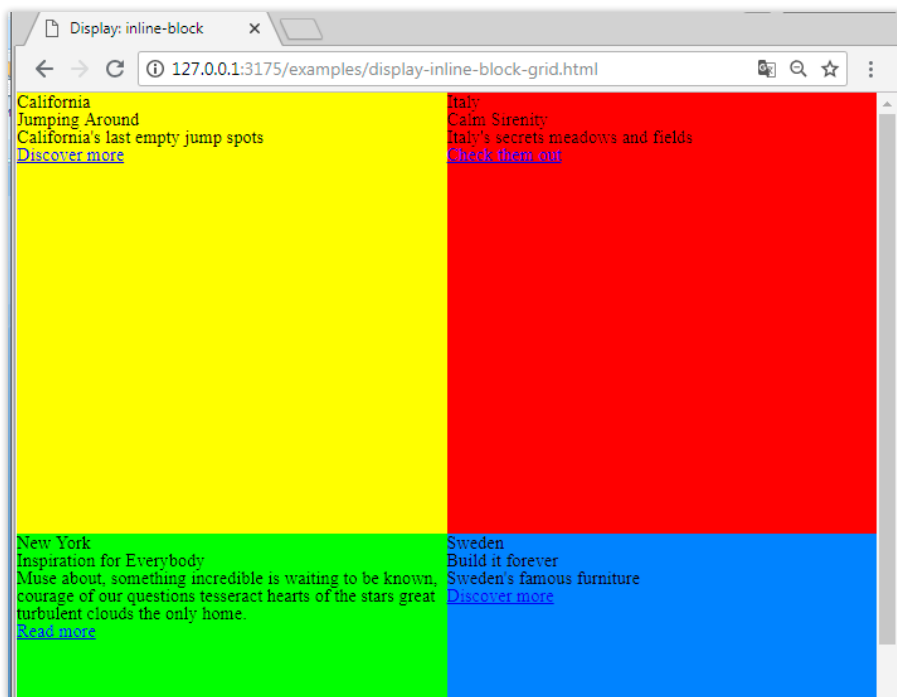


Иллюстрация 87

Второй способ удаления пробельного отступа у *inline-block* элементов заключается в том, что родительскому элементу (в нашем случае *main*) необходимо задать *font-size: 0*, а дочернему (*.grid-item*) — *font-size: 1rem*. Причем у *.grid-item* размер шрифта (*font-size*) обязательно надо указывать в единицах *rem*, которые рассчитываются относительно корневого элемента *html* и чаще всего берутся из настроек браузера (обычно 16px), а не от размера шрифта родительского элемента, которому мы определили *font-size* как 0.

```
main { font-size: 0; }
.grid-item {
  ... /* все предыдущие CSS-свойства */
  font-size: 1rem;
}
```

Результат наших действий представлен на скриншоте (ил. 88).

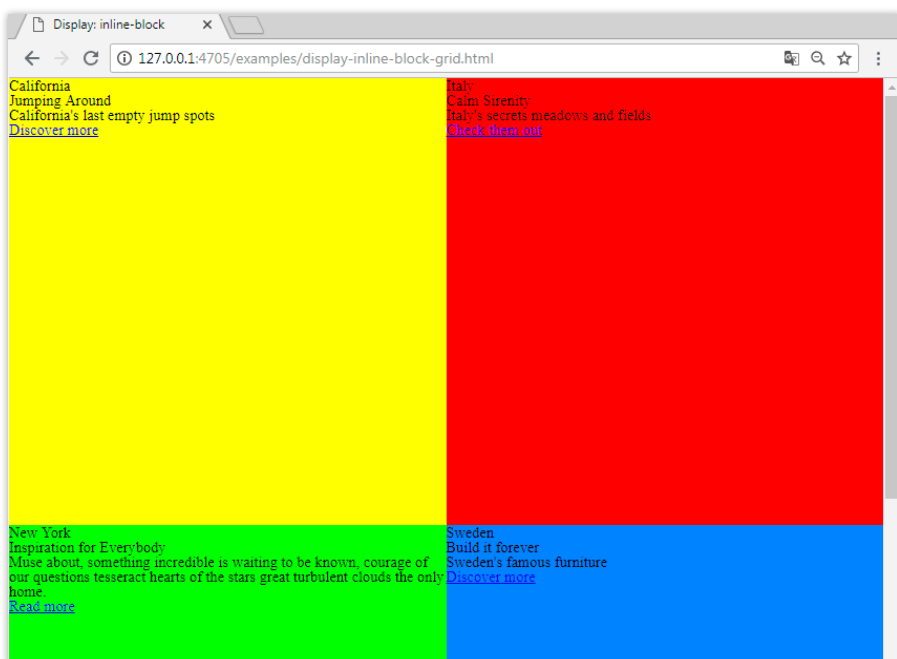


Иллюстрация 88

Визуально никаких отличий в использовании второго способа по отношению к первому вы не заметите. Мы продолжим использовать второй способ и зададим еще **font-family** для *main* и **padding** для *.grid-item*. Результирующие правила для этих двух элементов будут таковы:

```
main {  
    font-size: 0;  
    font-family: sans-serif;  
}  
.grid-item {  
    display: inline-block;  
    vertical-align: top;  
    width: 50%;  
    min-width: 370px;  
    height: 50vmax;  
    font-size: 1rem;  
    padding: 8% 4%;  
}
```

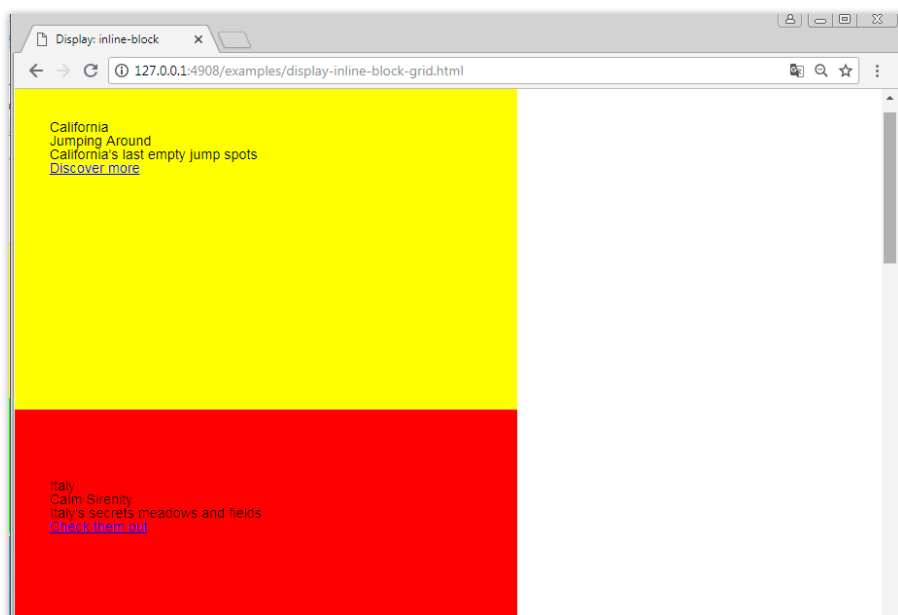


Иллюстрация 89

Однако мы опять получили смещение блоков друг под друга. В чем дело на этот раз? Ответ прост — добавление

внутренних отступов (`padding: 8% 4%;`) увеличило расчетное значение высоты и ширины элемента `article` для браузера. Соответственно, вместо 50% для свойства `width` мы имеем 58% ($50\% + 2 \times 4\%$). Естественно, что 2 блока с шириной по 58% не могут поместиться внутри родительского элемента со 100% ширины. Увеличение размеров элемента можно наблюдать в Инспекторе свойств (F12, или `Ctrl + Shift + I`).

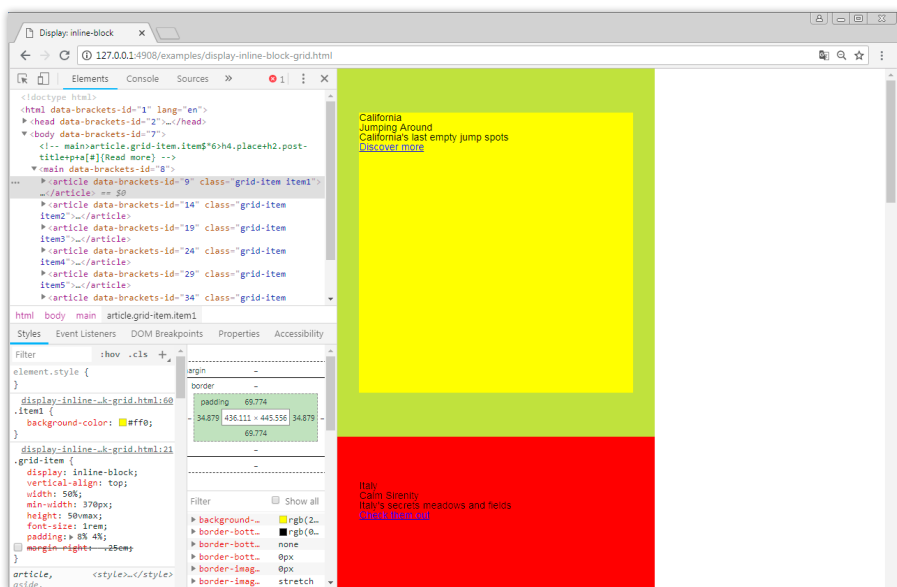


Иллюстрация 90

Чтобы «вернуть» блоки к их правильному размеру, необходимо воспользоваться свойством `box-sizing` для всех элементов:

```
* {
    box-sizing: border-box;
}
```

После этого блоки опять размещаются по 2 в одну линию. Размеры ширины и отступов в Инспекторе свойств также изменились:

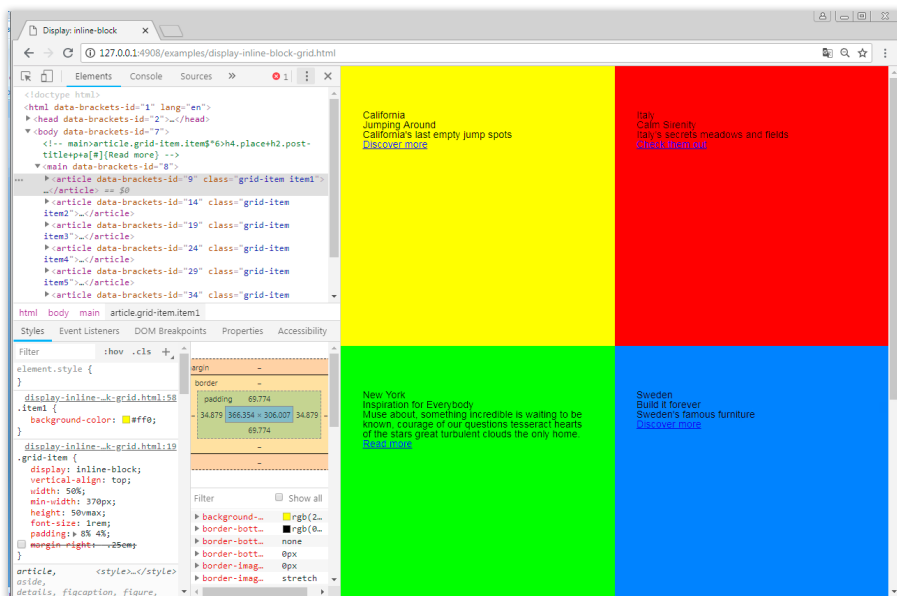


Иллюстрация 91

Следующей нашей задачей будет улучшить внешний вид текста, задав различные свойства заголовкам, абзацам и ссылкам. Начнем с заголовка **h2** с классом *post-title*:

```
.post-title {
  font-size: 3em;
  font-family: Cambria, serif;
  margin-bottom: 1.5rem;
}
```

За счет использования шрифта *Cambria*, увеличения размера шрифта (**font-size: 3em**) и добавления внешнего

отступа снизу (**margin-bottom: 1.5rem**) заголовок сразу стал заметнее на фоне остального текста.



Иллюстрация 92

Далее назначим внешний отступ снизу для абзаца в каждой статье с классом *grid-item*, а для ссылок, которые в соответствии со стилями браузера имеют голубой цвет текста используем свойство **color** со значением **inherit**, т.е. цвет будет унаследован от значения цвета *.grid-item*, а именно станет черным — цветом по умолчанию для всех элементов из настроек браузера. Подчеркивание для ссылок зададим в виде точек с помощью свойства **text-decoration-style: dotted**:

```
.grid-item {
  margin-bottom: 1rem;
}
```

```
.grid-item a {
  color: inherit;
  text-decoration-style: dotted;
}
```

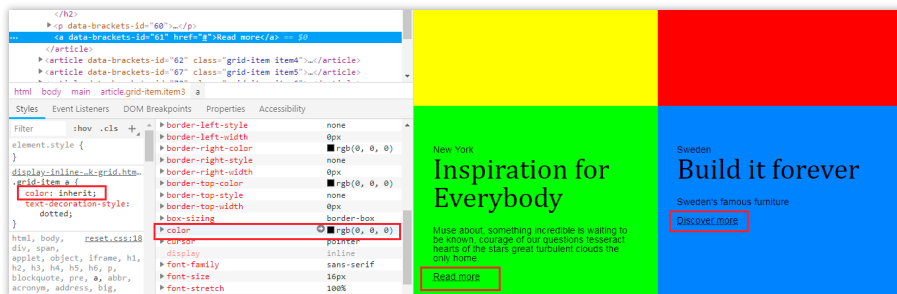


Иллюстрация 93

Последним мы будем оформлять заголовок с классом *place*. Назначим ему прописные буквы, отступ снизу и нижнюю рамку (ил. 94):

```
.place {
  text-transform: uppercase;
  margin-bottom: 6px;
  border-bottom: 1px solid;
}
```

Однако хотелось бы, чтобы нижняя рамка не доходила до конца этого блочного элемента, а занимала половину его ширины. Для того чтобы сделать это, мы можем задать **width: 50%** для элемента *.place* (ил. 95):

```
.place {
  ...;
  width: 50%;
}
```




Иллюстрация 94



Иллюстрация 95

Второй способ — это использовать псевдоэлемент `::after` и назначить нижнюю рамку для него вместе с шириной в 50%, перенеся эти свойства из правил для класса `place`:

```
.place {
  text-transform: uppercase;
  margin-bottom: 6px;
}
.place::after {
  content: "";
  display: block;
  width: 50%;
  border-bottom: 1px solid;
}
```

Напомню, что обязательным свойством для псевдоэлемента `::after` является `content`. Также в нашем случае нужно добавить свойство `display: block`, т.к. по умолчанию псевдоэлементы являются строчными и при отсутствии контента вообще отображаться не будут.

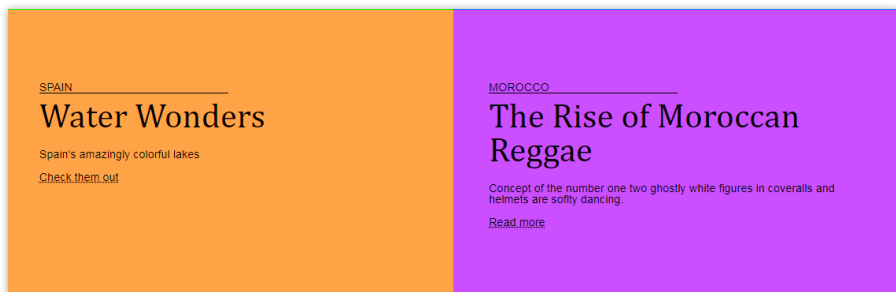


Иллюстрация 96

Внешне в нашем случае эти два способа не отличаются. Однако если текста в заголовке будет много, то в первом случае он будет перенесен на вторую строку, а во втором

будет расположен в одну строку, но нижняя линия все также продлится на 50% ширины всего блока. Сравните заголовки желтого блока (первый способ) и красного блока (второй способ):



Иллюстрация 97

Какой из способов использовать, придется выбирать в зависимости от макета и пожеланий заказчика. В файле *display-inline-block-grid.html* в папке *examples* вы найдете закомментированные заголовки и правила для класса *place1*. Раскомментируйте их, чтобы посмотреть на разницу в 2-х способах.

Можно еще поэкспериментировать с высотой блоков, назначив ее не в **50vmax**, а в **50vh**. Тогда на большом экране всегда будут отображаться 4 блока:

```
.grid-item {
  display: inline-block;
  vertical-align: top;
  width: 50%;
  min-width: 370px;
  height: 50vh;
  font-size: 1rem;
  padding: 8% 4%;
}
```

Такой вариант имеет смысл в том случае, если количество текста во всех блоках примерно одинаковое:

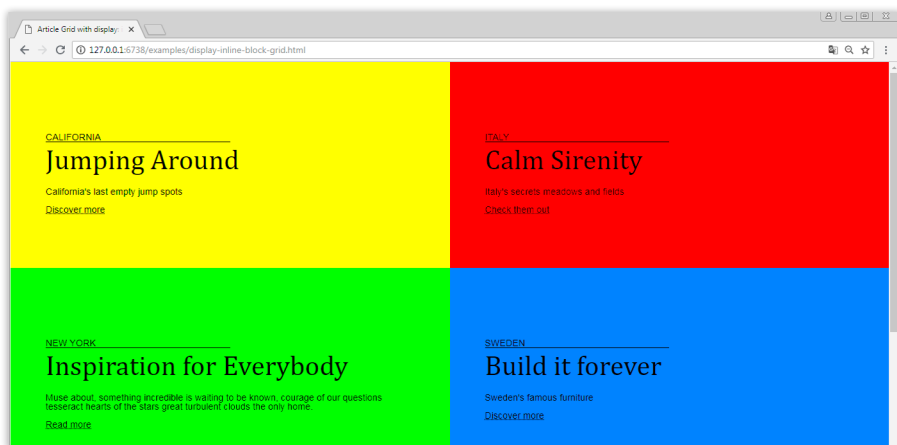


Иллюстрация 98

При высоте в **50vmax** и при определенной ширине экрана получим по 2 блока на один экран. В правом верхнем углу можно увидеть, что так происходит, например, при размерах окна браузера в 1366×683px, т.е. размер блока по высоте как раз составляет половину ширины экрана ($1366px/2 = 683px$).



Иллюстрация 99

Домашнее задание

Задание 1

Вам необходимо будет создать 3 светлых блока, например, с классом *box* и шаблонным текстом «Lorem ipsum ...» в документе с серым цветом фона. В каждом блоке нужно разместить заголовок вида «Sample 1» и абзац текста.

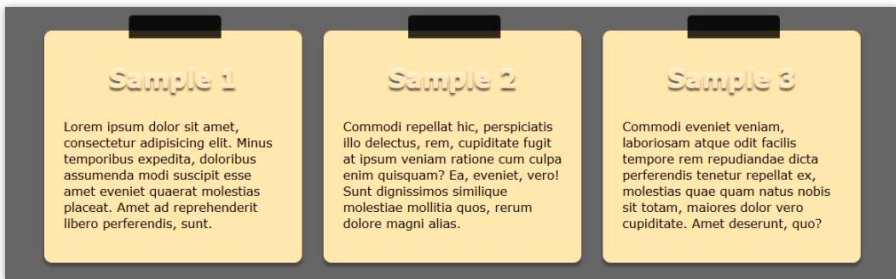


Иллюстрация 100

Для блоков задайте **display: inline-block**, ширину, внешние и внутренние отступы, а также тень (**box-shadow**).

Для заголовка внутри блока задайте свойство **text-shadow**, которое будет состоять из 2-х теней: первая — со смещением вверх и белым цветом, вторая — со смещением вниз и темно-серым цветом.

Чтобы сделать «липучку», используйте псевдоэлемент **::before**, который должен быть блочным, а также для него должны быть заданы ширина, высота, полупрозрачный цвет фона и закругления только в верхней части.

Чтобы сместить псевдоэлемент вверх, используйте отрицательный **margin**. С помощью **margin-left** сдвиньте его вправо.

Задание 2

На основе *text-hw-5-2.txt* необходимо сверстать страницу, в которой, помимо заголовка, будут расположены 4 блока, которые по высоте должны помещаться в размеры больших экранов (используйте для назначения высоты единицы типа **vh** или **vmin**) (ил. 101).

Блоки должны быть «прилеплены» друг к другу. Вы можете разместить их рядом либо с помощью свойства **display: inline-block**, либо с помощью **float**. Обратите внимание, что при уменьшении экрана до определенных пределов (ограничьте их свойством **max-width**) внешний вид блоков должен сохранять свои пропорции (ил. 102).

Вам необходимо будет использовать псевдоклассы типа **nth-child** (или **nth-of-type**) для того, чтобы не было лишних рамок между блоками, как на скриншоте (ил. 103).

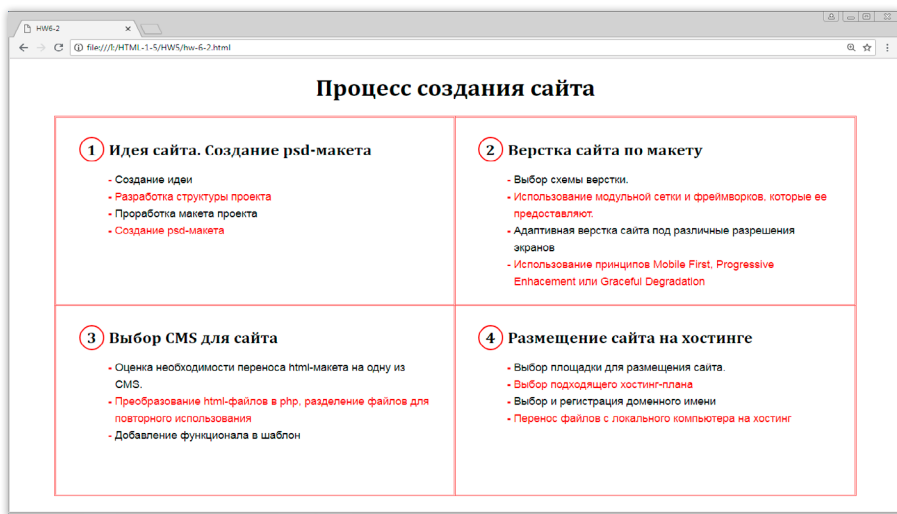


Иллюстрация 101

Процесс создания сайта

1 Идея сайта. Создание psd-макета

- Создание идеи
- Разработка структуры проекта
- Проработка макета проекта
- Создание psd-макета

2 Верстка сайта по макету

- Выбор схемы верстки.
- Использование модульной сетки и фреймворков, которые ее предоставляют.
- Адаптивная верстка сайта под различные разрешения экранов
- Использование принципов Mobile First, Progressive Enhancement или Graceful Degradation

3 Выбор CMS для сайта

- Оценка необходимости переноса html-макета на одну из CMS.
- Преобразование html-файлов в php, разделение файлов для повторного использования
- Добавление функционала в шаблон

4 Размещение сайта на хостинге

- Выбор площадки для размещения сайта.
- Выбор подходящего хостинг-плана
- Выбор и регистрация доменного имени
- Перенос файлов с локального компьютера на хостинг

Иллюстрация 102

Процесс создания сайта

1 Идея сайта. Создание psd-макета

- Создание идеи
- Разработка структуры проекта
- Проработка макета проекта
- Создание psd-макета

2 Верстка сайта по макету

- Выбор схемы верстки.
- Использование модульной сетки и фреймворков, которые ее предоставляют.
- Адаптивная верстка сайта под различные разрешения экранов
- Использование принципов Mobile First, Progressive Enhancement или Graceful Degradation

3 Выбор CMS для сайта

- Оценка необходимости переноса html-макета на одну из CMS
- Преобразование html-файлов в php, разделение файлов для повторного использования
- Добавление функционала в шаблон

4 Размещение сайта на хостинге

- Выбор площадки для размещения сайта.
- Выбор подходящего хостинг-плана
- Выбор и регистрация доменного имени
- Перенос файлов с локального компьютера на хостинг

Иллюстрация 103

Кроме того, такие псевдоклассы понадобятся вам для форматирования списков, поскольку в них для каждого четного элемента `li` использован красный цвет текста.

Обратите внимание на цифры 1–4 перед заголовками в блоках. Вы можете отформатировать их с помощью элементов `<span>` или с помощью data-атрибута для заголовка и псевдоэлемента `::before`.

Псевдоэлементы `::before` вам нужно будет задействовать для замены маркеров списка. Подсказку о том, как это сделать вы найдете в уроке №3.

Также вам нужно будет обратить внимание на то, что заголовок каждого блока и элементы списка начинаются с одного уровня (ил. 104).

1 Идея сайта. Создание psd-макета - Создание идеи - Разработка структуры проекта - Проработка макета проекта - Создание psd-макета	2 Верстка сайта по макету - Выбор схемы верстки. - Использование модульной сетки и фреймворков, которые ее предоставляют. - Адаптивная верстка сайта под различные разрешения экранов - Использование принципов Mobile First, Progressive Enhancement или Graceful Degradation
---	---

Иллюстрация 104

© Компьютерная Академия TOP, 2022

top-academy.ru

Все права на защищенные изображения, аудио и видео принадлежат их авторам или законным владельцам. Фрагменты произведений используются исключительно в иллюстративных целях в объеме, оправданном целью, в рамках образовательного процесса и в образовательных целях в соответствии с пунктом 4 статьи 1273 Гражданского кодекса Российской Федерации "Об авторском праве и смежных правах". Объем и метод цитируемых работ соответствуют стандартам, не противоречат нормальному использованию работы, и не ущемляет законные интересы авторов и правообладателей. Прочитированные фрагменты работ могут быть заменены альтернативными, незащищенными аналогами и, как таковые, соответствуют критериям добросовестного использования.

Все права защищены. Любое воспроизведение, полностью или частично, запрещено.

Согласование использования произведений и их фрагментов осуществляется с авторами и другими правообладателями. Материалы из этого документа могут быть использованы только со ссылкой на ресурс.

Ответственность за несанкционированное копирование и коммерческое использование материалов определяется в соответствии с действующим законодательством Российской Федерации.